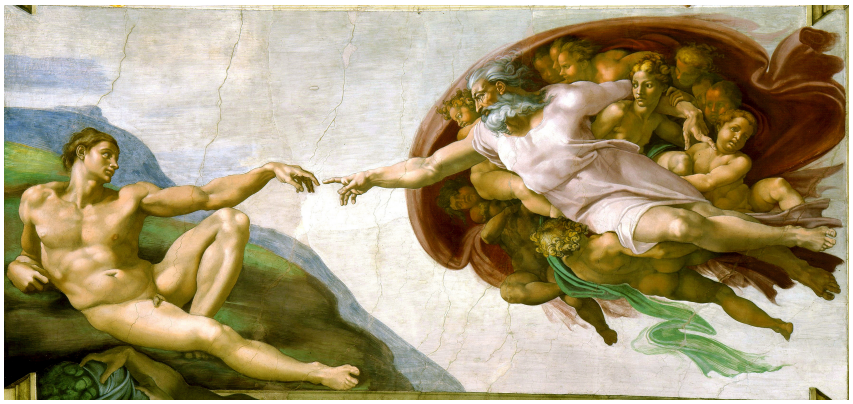
The GNU Java Training Wheels programming language for a simplified version of the Java programming language



Part 1/3 of a Ph.D. Thesis

By Davin Pearson

Eleventh Edition

The GNU Java Training Wheels programming language for a simplified version of the Java programming language.

Part 1/3 of a A Ph.D. thesis

January 17, 2019

Abstract

This book is about how to add a preprocessor to the Java language to turbo-charge its performance and to create a new programming language called GNU Java Training Wheels *or* J.T.W. *for short. Both expressiveness and efficiency can be improved using preprocessor languages. J.T.W. has been created specifically for novice Java programmers who want to learn Java. In particular Pascal-style* **begin** *. . .* **end** *constructs are supported instead of Java's* { *. . .* } *construct, which makes J.T.W. code much more readable than the equivalent Java code. J.T.W. translates to Java in a natural and straightforward manner so it is easy for J.T.W. programmers to learn Java. J.T.W. is supported by easy to understand error messages so it is easy to debug J.T.W. code. For many reasons you might prefer to code in J.T.W. rather than Java. Experienced programmers will find J.T.W. useful too. Emacs Lisp is used as the preprocessor for the Java and C++ languages because it is powerful enough for my needs and it is free software. That is to say free as in free speech and not free beer. Lisp is a higher level language than Java and is powerful enough to render obsolete blocks of tiresome repetitive boilerplate code that dominates code written in Java. A small collection of* **d-defmacro***s have been provided for you to deploy in your client code. If you are especially clever, you can write your own Emacs Lisp* **d-defmacro***s to replace blocks of tiresome repetitive boilerplate code in Java. The idea for eliminating tiresome repetitive boilerplate code comes from Peter Seibel's 2005 book* [Sei05] Practical Common Lisp *which devotes an entire chapter (chapter 9) to eliminating tiresome repetitive boilerplate code from Common Lisp code.*

Published by lulu.com in association with davinpearson.com.

ISBN: 978-0-244-348786

[1] www.gnu.org/copyleft/fdl.html

For Dorothy

Contents

Preface

Preface to the eleventh edition

Split my book from one book into two separate books, *The Java Training Wheels programming language* and *Building C++ Preprocessors: Using Lisp++ for Efficient and Expressive Programing*

Preface to the tenth edition

Removed the C++ source code for the `libd` library because C++ is not supported by this version (and future versions) Added a new section §**??** called *A solution to the first problem.* Added a new section **??** called *Proof of concept 1: A small collection of d-defmacros for your use in your Lisp++ client code.* Also fontified all occurrences of `private_foo` in the face "`prvt`", short for *private.*

Preface to the ninth edition

Fixed numerous typographical errors. Changed the link of my large files links from

`davinpearson.com/binaries/large-files-links.html`

to

`davinpearson.com/binaries`

so that uploads to this website are displayed by default without the need to update the file `large-files-links.html`.

Preface to the eighth edition

Changed the save names for classes that begin with an initial capital letter. This overcomes Microsoft Windows' limitation in its filenames in how it cannot have two files with the same name, only different in case, e.g. `foo` and `Foo`. Therefore a **class** `X` will now reside in files called `_X.lisp++` and will be built into C++ source files `_X.h++`, `_X.ch++` and `_X.c++`. That way a **class** called `x` can reside in a file called `x.lisp++` and will be built into files called `x.h++`, `x.ch++` and `x.c++` and Windows won't complain about three pairs of files different only in case. Actually

instead of complaining, Windows silently overwrites one of each pair of files with the other, which is hardly ideal behaviour. This scheme of things works equally well in GNU/Linux but is superfluous in this case.

Preface to the seventh edition

Added syntax highlighting to the following textual elements:

NOTE: I am a note

COOL: I am a cool note

and similar textual elements. Added the following target to the manual's Makefile in §2.11.3 that was missing from earlier editions:

```
build-class-db:
        @echo STRINGBGFG("* Stage 0 :  Building class database")
        emacs --batch --eval STRINGBGFG("(setq dir \"$(PREFIX)/share/emacs/site-lisp/dlisp/\")") \
--load $(PREFIX)/share/emacs/site-lisp/dlisp/jtw-build-class-db.el --funcall doit

clean: build-class-db
```

Added section §?? on installing a C/C++ compiler.

Preface to the sixth edition

Put back sections §?? and §?? that were accidentally removed from the previous edition. In §2.16.4 removed the fontification of the word **main** → main. Also changed **begin**{**enumerate**} ... **end**{**enumerate**} → **begin**{**itemize**} ... **end**{**itemize**} in section §2.10. Centralised the diagrams in Figures ??, ?? and ??.

Preface to the fifth edition

Upped the number of lines of code written from 53,000 → 54,000. Moved *An idiom for* **constructor***s* from §?? to §2. Also updated the code to reflect this change. Expanded the section in §??. Removed the section *Debugging crappyness of Lisp++* since it no longer applies.

Preface to the fourth edition

Added a new section *Virtual Methods*, see §??. Added a new section *Run Time Type Inquiry*, see §??. Clipped extra long lines in the code listing in §2.7.2. Renamed **method**s in §?? from **x_method1** → **foo_method1** etc. Corrected the following hyperlink in §?? davinpearson.com/binaries/large-files- → davinpearson.com/binaries/large-files-links.html Improved the diagram in Figure 2.1.

Preface to the third edition

Added support for inline **function**s and **method**s and documentation of the `cinline` keyword. See §?? for more information. Fixed the following bug in the documentation. See §2.16.4.

A → pkg.inner.A

Upped the lines of Emacs Lisp source code written count from 41,000 → 53,000 lines of code. I now count experimental code as well as actively used code to get the higher value for the number of lines of code written. This bumped up the number of lines of code by over 6,000.

Preface to the second edition

Removed the extraneous large source code file: `Othello.lisp++` (1,000+ lines of code) from the first edition of my book. Updated the lines of Emacs Lisp source code written count from 38,000 → 41,000 lines of code.

Preface to the first edition

Wrote this book using the LaTeX document-markup system, specifically `pdfTex` Version 3.1415926-2.5-1.40.14 (TeX Live 2013/Debian). Also used the program `xfig` for drawing diagrams. Used the following Emacs Lisp code for syntax highlighting the various code language buffers, using LaTeX's **color**{*color name*}{*text to colourise*} and **colorbox**{*color name*}{*text to colourise*}.

davin.50webs.com/research/2010/d-latexize7.el.html

Executed `d-latexize.el` by issuing the following shell command:

```
emacs --batch --eval STRINGBGFG("(setq *target* \"/path/to/filename\")") ↩
--load $(PREFIX)/share/emacs/site-lisp/dlisp/d-latexize7.el --funcall doit
```

where ***/path/to/filename*** is the name of the file you want to include into your LaTeX sources. In the above printout, note the use of the symbol ↩ to refer to a line of code that has been clipped to fit onto the page. Note that `$(PREFIX)` is set by default to `/usr/` under GNU/Linux or `c:/java-training-wheels/` under M.S. Windows. Ran the LaTeX fontification engine on itself to print out the following printout. Note the use of GNU m4 to provide logic for the printout:

```
// BEGIN FILE: ../m4-emacs-pretty-print-latex2.m4
m4_define([m4_emacs_pretty_print_latex],m4_dnl
[\noindent{}m4_ifelse(-1,m4_regexp($1,el),{\color{comm}{//}},{\color{comm}{;;}}) m4_dnl
{\bf\colorbox{begin-code-bg}{\color{begin-code-fg}{{\bf B}EGIN FILE:}}} m4_dnl
{\bf\color{black}{m4_patsubst(m4_patsubst($1,_,\\_),~,\\~{})}}m4_dnl
m4_syscmd(emacs --batch --eval "(setq *target* \"$1\")" ↩
--eval "(setq load-path (cons \"~/dlisp\" load-path))" ↩
--load ~/dlisp/d-latexize7.el --debug-init ↩
--funcall doit)
m4_esyscmd(cat $1.tex) m4_dnl
m4_ifelse(-1,m4_regexp($1,el),{\color{comm}{//}},{\color{comm}{;;}}) m4_dnl
{\bf\colorbox{begin-code-bg}{\color{begin-code-fg}{{\bf E}ND FILE:}}}\hspace{3.76mm} m4_dnl
{\bf\color{black}{m4_patsubst(m4_patsubst($1,_,\\_),~,\\~{})}}
m4_syscmd(rm -f $1.tex) m4_dnl
])
// END FILE: ../m4-emacs-pretty-print-latex2.m4
```

This macro is called like so:

```
m4_begin_indent
m4_emacs_pretty_print_latex(/path1/to/File.java) m4_dnl java-mode file
m4_emacs_pretty_print_latex(/path2/to/File.jtw) m4_dnl jtw-mode file
m4_emacs_pretty_print_latex(/path3/to/file.cc) m4_dnl c++-mode file
m4_emacs_pretty_print_latex(/path4/to/file.c++) m4_dnl c++-mode file
m4_emacs_pretty_print_latex(/path5/to/file.el) m4_dnl emacs-lisp-mode file
```

```
m4_emacs_pretty_print_latex(/path6/to/file.lisp++) m4_dnl lisp++-mode file
m4_end_indent
```

Where `m4_begin_indent` and `m4_end_indent` are defined like so:

```
m4_define([m4_begin_indent],[m4_dnl
\begin{quote} m4_dnl
\begin{tt} m4_dnl
\begin{footnotesize} m4_dnl
m4_changequote(,) m4_dnl Turns m4 quotes off.
])
```

and like so: I get by with a little help from my friends. It's getting better all the time.

lucy in the sky with diamonds

you're such a lovely audience

```
m4_define([m4_end_indent],[m4_dnl
\end{footnotesize} m4_dnl
\end{tt} m4_dnl
\end{quote} m4_dnl
m4_changequote(,) m4_dnl Turns m4 quotes off
m4_changequote([,]) m4_dnl Changes m4 quotes back to [ ... ]
])
```

Chapter 1

Introduction

I get by with a little help from my friends..

sexy rexy

you're such a lovely audience

This book is about how to add a preprocessor to the Java language to turbo-charge its performance. Both expressiveness and efficiency can be improved using preprocessor languages. The preprocessor language is *J.T.W.*. J.T.W stands for *Java Training Wheels*, and is intended for computer programming novices. The name Java Training Wheels was the outcome of an email conversation with Dr. Richard Stallman][1], the President of the *Free Software Foundation*][2] and founder of the *GNU Project*][3], creator of *GNU Emacs*][4], the *GCC compiler*][5], and the *GNU Debugger*][6] which ultimately resulted in the *GNU/Linux*] [7] operating system.

Since August 2016, J.T.W. has been accepted by Richard Stallman for inclusion into the Free Software Foundation's repository of Free software, so it is now known by the slightly longer name *GNU Java Training Wheels*. Visit the following Web page on GNU's Website for more information:

www.gnu.org/software/jtw

J.T.W. for example allows programmers to learn programming within an environment that resembles *Pascal* and *BASIC*.

A small collection of **d-defmacros** have been written for you to deploy in your client code. If you are especially clever, then you can write your own **defmacros** to eliminate tiresome repetitive blocks of "boilerplate" code in Java. See §2.7.1 for how to add your own code to J.T.W.

As further proofs of concept for J.T.W. a **superfor** macro (see §2.7.2) is presented (much like the **for** loop construct in BASIC), as well as a file inclusion system (see §2.7.3).

When I first learned the C programming language I was impressed by the power of its preprocessor. Now in the twenty-first century, the C/C++ preprocessor seems like a remnant from the dinosaur age with its lack of support for `#define`s with multiple **template** arguments and the need for excessive backslashes to include blocks of code. Also I believe that the C/C++ preprocessor is not so-called *Turing complete*, which means that its computational power is severely limited. Emacs' suitability for both preprocessing and editing preprocessor code will soon be demonstrated to you the reader, if you will bare with me I will take you on a tour through some existing languages and show you how their performance can be turbo-charged.

[1] stallman.org
[2] fsf.org
[3] gnu.org
[4] en.wikipedia.org/wiki/GNU_Emacs
[5] en.wikipedia.org/wiki/GNU_Compiler_Collection
[6] en.wikipedia.org/wiki/GNU_Debugger
[7] en.wikipedia.org/wiki/GNU/Linux

After learning the C and C++ language, I learned the similar GNU m4 programming language[8] which is similar to the C/C++ preprocessor only more powerful, and used it to build a large (over 500 page) Website at

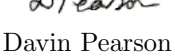
davin.50webs.com

Sometime in between learning C++ and m4 I learned Java and used my knowledge of it to tutor Stage I students in the language. Then I invented the J.T.W. programming language which is intended for novices to help them to learn the Java language. I originally used m4 to compile J.T.W. source code into Java code. It was then that I learned about m4's limitations, specifically how m4 operates on strings when it should leave them alone unchanged. More on this later.

I considered using Flex to compile J.T.W. into Java code but for simplicity I chose the slower but simpler and more powerful technique of using GNU Emacs as a preprocessor. Specifically, Emacs' *batch mode* is used to compile J.T.W. into Java code. The batch mode code is written in *Emacs Lisp* (or *Elisp* for short at the risk of confusion with an older unrelated language called Elisp), the extension language for the GNU Emacs editor. Emacs is available but not compulsory to be used as an editor. The main advantage of using Emacs as an editor as well as a preprocessor is that it allows for **syntax highlighting** of J.T.W. constructs or whatever constructs your language uses for the general case of adding a preprocessor language to your favourite language. Also Emacs provides correct automatic indentation of J.T.W. code.

The J.T.W. programming language is subject to the GNU General Public License for maximum freedom of extension. Therefore this program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See Chapter §3 for the license agreement.

Enjoy reading my book!



Davin Pearson
Christchurch
New Zealand
January 17, 2019

[8] en.wikipedia.org/wiki/GNU_m4

Chapter 2

The J.T.W. language

2.1 Why learn to use J.T.W.?

The first part of this book presents a new programming language called *J.T.W.*, short for *Java Training Wheels* for the sole purpose of making it easier to learn to program in Java. The J.T.W. language has a similar syntax to Delphi, Pascal, BASIC and JavaScript and therefore learning J.T.W. before or while learning Java provides a less steep learning curve than learning Java from scratch. For many reasons you might even prefer to program in J.T.W. rather than Java. Here is why you should learn J.T.W. before or while learning Java:

- The J.T.W. language is supported by a parser that troubleshoots problematic J.T.W. code with clear error messages.
- The J.T.W. language compiles to Java in a natural and straightforward way so it is easy to learn Java once you know J.T.W. See Figure 2.1 for a comparison of the J.T.W. and Java build processes.
- Pascal-style **begin** ... **end** constructs are supported instead of C-style { ... } constructs which is more sensible especially for novices.
- A simple syntax for the **main** **function**: **beginMain** ... **endMain** rather than the rather cumbersome: **public static** void **main**(String[] args) { ... }.
- Class **variable**s, **property**s, **function**s, **method**s and **constructor**s are declared as such much like Delphi which makes your code look clearer. In particular there are new keywords **classVar**, **property**, **function**, **method** and **constructor**.
- The Delphi/Pascal/JavaScript keyword **var** for clearer local **variable**s.
- The Pascal/BASIC keyword **then** for clearer **if** statements.
- The BASIC keywords **and** and **or** rather than Java's rather cumbersome: `&&` and `||`
- As proof of concept, a **superfor** macro is presented for enhanced BASIC-style **for** loops.
- As proof of concept, file inclusion is supported so that you can spread a **class** across several files. Natural divisions are **method**s. Different **method**s can be placed in different source files for those situations where **method**s become large and unwieldy.

NEW! J.T.W. Version 1.1 supports packages

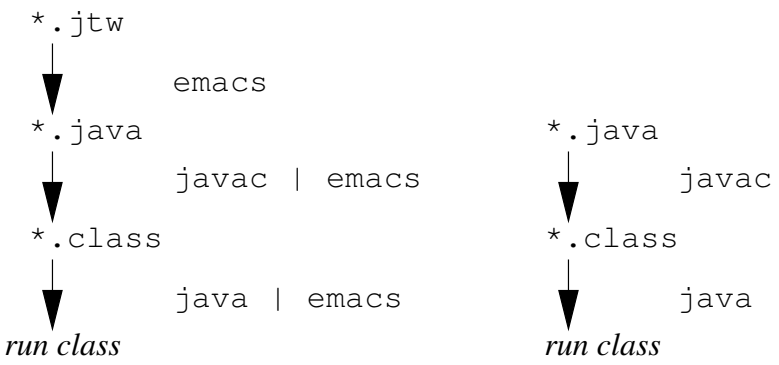


Figure 2.1: Above left is J.T.W.'s build process. Above right is Java's build process. **NOTE:** the vertical bar | represents a piping of the output of the first command into the input of the second command. In the case of Emacs, its *batch mode* rather than *interactive mode* is used in the build process. See §2.11.3 for the *GNU Makefile* for the details of this build process.

2.2 GNU Emacs as a development environment

2.2.1 Why use GNU Emacs as your development environment?

GNU Emacs is the most powerful editor in existence. Most of the Emacs source code is written in a high level language called *Emacs Lisp* or *Elisp* for short. Therefore it is much easier to add customizations than for any other program written in a lower level language such as C or C++. Code can be easily written so that Emacs can host any language you care to use. For J.T.W. the code has already been written for you in the form of `jtw-mode.el`. You can choose to use Emacs with Davin Pearson's customizations or Emacs with just Davin's `jtw-mode.el`. It is recommended that you use Emacs with all of Davin's customizations (also known as *Davin's Full Version of GNU Emacs*) for maximal editing effectiveness. See the following website `www.emacsrocks.com` for some cool stuff that Emacs can do.

2.2.2 Installing GNU Emacs

Installing GNU Emacs on Windows P.C.'s

1. First you need to download `emacs-25.2-i686.zip` or a later version from GNU's Website: `ftp://ftp.gnu.org/pub/gnu/emacs/windows`]. The file size is approximately 92 megabytes, about the size of twelve *MP3* songs. The download time should a few minutes on Broadband Internet.

2. Then you need to unzip the archive to your `c:/Program Files` folder.

3. Then you need to set the `HOME` environment variable to a sensible value for your system. If you have only one hard drive, then the most appropriate value for `HOME` is `c:/home`. If you do not set the `HOME` variable, it will default to `c:/` but the problem with this is that the **root** directory of your hard drive will be cluttered with a whole bunch of files beginning with the period character (.), eg. `.*`. Here is how you should go about achieving this:

   (a) Firstly minimise any open windows.

   (b) Press **Windows E** to open *Windows Explorer*.

   (c) Right click on *This P.C.* or *My Computer*, depending on what version of Windows you are running.

   (d) Click on *Properties* and then click on *Advanced*.

(e) Click on *Environment Variables*.

(f) In the *User variables* or *System variables* section, if there already is a value for the HOME variable, then either keep it or change it to a sensible valuesuch as `c:\home`.

(g) To change it, click on HOME and then click *Edit*.

(h) When you have finished editing it then click on *OK* Keep pressing *OK* until you have no windows left to close.

4. In Windows Explorer, click on the `c:` drive, then `Program Files` then `emacs-25.0.95` (or whatever version of Emacs that you have installed on your system), then `bin` then `addpm.exe` to add a button to copy the Start Emacs button to your Desktop.

5. In the folder pointed to by the `HOME` variable, create a file called `.emacs` and save it to disk. You can use *Notepad* to create such a file. To open Notepad, click on the *Start* button, then *All Programs*, then *Accessories*, then *Notepad*.

2.3 Installing the installer module for `c++2lisp++2c++`

To install `c++2lisp++2c++` and, optionally Davin's Full Version of GNU Emacs, follow the following instructions:

1. Untar the tarball `preprocessors-YYYYmmdd-HHMMSS.tar.gz`.

2. Change directory to the following directory: `~/preprocessors`, and run the following command under M.S. Windows

   ```
   bash install username ENTER
   ```

   Note that under GNU/Linux you will need to be logged in as the `root` user. To achieve this, simply wrap the above command with `su ...exit` like so:

   ```
   su
   bash install username ENTER
   exit ENTER
   ```

   Note that you will be prompted for the `root` password.

3. Note that under M.S. Windows you will need to have the program `bash.exe` installed on your system. You can install this program from Cygwin[1]. It should be already installed on GNU/Linux systems. When running the install script, you will be asked for the location of the prefix directory, the destination directory for your J.T.W. files, and whether or not to install Davin's Full Version of GNU Emacs.

4. If you have the program `yes` installed (as will be the case if you are running GNU/Linux or Cygwin[1]) then you can run the installer module with all of the default settings by issuing the following command. Note that the default setting is **not** to install Davin's Full Version of GNU Emacs. Use the following command under Windows:

   ```
   yes | bash install ENTER
   ```

   or the following command user GNU/Linux:

   ```
   su yes | bash install ENTER exit
   ```

[1] Visit the following Website: www.cygwin.com for the program `setup.exe` which will install this program (and others too).

Installing GNU Emacs on GNU/Linux systems

In *GNU/Linux* systems that derive from *Debian*,[2] all you need to do is to type the following command from your Bash prompt:

```
su
apt-get install emacs25 ENTER
exit ENTER
```

To execute this command, you will be prompted for the **root** password.

Installing `bash`, `grep`, `make` and `sed`

To run J.T.W. files you need to have `bash`, `grep`, `make` and `sed` installed on your system, which you can install yourself if you are using cygwin. If you are running a GNU/Linux system these commands will already be installed. If you are using Cygwin under M.S. Windows then you can download the executables using the already-mentioned command `setup.exe`

Under GNU/Linux systems that derive from Debian, execute the following command

```
su ENTER
apt-get install package ENTER
```

where *`package`* is a name of the package that you want to install. Note that you will be prompted for the **root** password.

2.3.1 Uninstalling `c++2lisp++2c++`

To uninstall `c++2lisp++2c++`, you need to issue following command. Note that you will be prompted for the `root` password:

```
su ENTER
bash uninstall username ENTER
exit ENTER
```

Assuming you have untarred the tarball `preprocessors-YYYYmmdd-HHMMSS.tar.gz` to the following folder: `~/preprocessors`, then you need to issue the following command to remove the files: `rm -fr ~/preprocessors`.

2.4 Introducing J.T.W. keywords

In §2.1 I explained how the J.T.W. keywords **begin** ... **end** replaces { ... }, and how the J.T.W. keywords **beginMain** ... **endMain** replaces **public static void main** (String[] args) { ... }. This section explains the rest of the J.T.W. keywords.

1. The J.T.W. keyword **var** makes it clearer whenever a new local variable is introduced. For example: The following J.T.W. code: **var** `int x = 123;` compiles to the following Java code: `int x = 123;`.

2. The J.T.W. keyword **classVar** is used to denote **class variable**s, also known in Java as **static variable**s.

[2] See the following link: www.debian.org/misc/children-distros for a list of GNU/Linux distributions which derive from Debian. The list includes `Ubuntu` (see ubuntu.com) and `Lubuntu` (see lubuntu.net) the flavour of GNU/Linux that I choose to use.

3. The J.T.W. keyword **property** is used to denote **property**s, also known as instance **variable**s.
4. The J.T.W. keyword **function** is used to denote **class method**s, those which in Java have the **static** keyword.
5. The J.T.W. keyword **constructor** is used to denote **constructor**s.
6. The J.T.W. keyword **method** is used to denote **method**s, those which in Java lack the **static** keyword.
7. The J.T.W. keyword **then** is used to make **if** statements more clear. For example: **if** `(abc)` **then begin** `...` **end** in J.T.W. compiles to **if** `(abc)` { `...` } in Java.
8. The **elseif** keyword for replacing **else if**.
9. The J.T.W. keywords **and** and **or** serve to replace Java's cumbersome `&&` and `||` for, respectively *logical and* and *logical or*.

2.5 Your first program

Traditionally the first program you write in any language is a program that does nothing but prints out "`Hello, world!`". Here is such a program in J.T.W. which belongs in a file called `MyFirstProgram.jtw`:

```
class MyFirstProgram
begin
   beginMain
      System.out.println(STRINGBGFG("Hello, world!"));
   endMain
end
```

Here is the same program as the above, after being compiled to Java. This code will reside in a file called **MyFirstProgram.java**.

```
class MyFirstProgram
{
   public static void main (String[] args)
   {
      System.out.println(STRINGBGFG("Hello, world!"));
   }
}
```

2.5.1 Building J.T.W. into Java and running class files

To build a single **class** file, you simply execute the command from your `~/jtw-tutorials` folder:

```
make build MyFirstProgram.run
```

which will build, in order, **MyFirstProgram.java**, `MyFirstProgram.class` before running

```
java -enableassertions MyFirstProgram
```

The purpose of the "`build`" target is to call the "`clean`" target which deletes all ***.java** and `*.class` files before building the target file. If you don't do this then `java` might run an old version of `*.class` files despite earlier errors in the build process. This is because the use of pipes in building and executing `*.class` files hides the return values of the programs `javac` and `java`. The `build` target is also useful also when compiling groups of `*.jtw` files.

2.6 J.T.W. Tutorials

These tutorials are also available on-line on my Website:

davin.50webs.com/J.T.W

The answers to the tutorials can be found at my Website above and are protected by passwords. For the passwords to the answers to the questions, see §2.17. To enter the passwords,scroll down to **Section 3: Answers to the tutorials**. and click on the hyperlink there.

- §2.6.1 Introducing **function***s*,*parameters*,*arguments*,*strings*,**System.out.println** and *comments* to give you enough basic J.T.W. to get you started.
- §2.6.2 **Tutorial 2: Introduction to programming in J.T.W.** Introducing chars,*the difference between* `==` *and* `=`,booleans,the **if** `(...)` **then** ... **elseif** `(...)` ... **elseif** `(...)` ... **else** ... construct,*local* **variable***s*,ints,the **superfor** construct and teaching you *how to call existing* **method***s of the string* **class** but not teaching you how to write your own **method**s until Tutorial 9.
- §2.6.3 **Tutorial 3: superfor loops and for loops.** Introducing **System.out.print** for printing without a trailing carriage return,revising loops that use the **superfor** construct,introducing doubles and revising ints and chars.
- §2.6.4 **Tutorial 4: Four looping constructs.** Other types of loops such as **while** and **do** ... **while**,and revising **if** (...) **then** ... **elseif** (...) ... **elseif** (...) ... **else** ... statements and **for** loops. Learning what is the best of these three looping constructs.
- §2.6.5 **Tutorial 5: A beer drinking song.** Using all of the J.T.W. constructs that you have learnt so far to rewrite a song to be more general-purpose.
- §2.6.6 **Tutorial 6: Class variables.** Introducing *class* **variable***s* which are different from **variable**s that are local to **function**s.
- §2.6.7 **Tutorial 7: Non-`Object` arrays.** Introducing *non-object arrays* that are either *single dimensional* or *multi dimensional* using *two different initialisation syntaxes* and introducing *function name overloading*.
- §2.6.8 **Tutorial 8: Accessing functions and class variables from another class.** Learning *how to access* **function***s and* **class variable***s from another* **class** and introducing *boolean arrays*.
- §2.6.9 **Tutorial 9: Mapping:**

  1. **class variable***s* → *instance* **variable***s* (which are better known as **property***s*),and
  2. **function***s* → **method***s*

  to allow for more than one object per **class**. This gives you the full power of O.O.P. *(Object Oriented Programming)* **class**es. Introducing *getter* **method***s* and *references* for accessing objects. Introducing the `null` keyword for representing *no object* and introducing the `toString` **method**,while explaining why this **method** is better than any other **method** or **property** for debugging your code.
- §2.6.10 **Tutorial 10: `Object` arrays.** Introducing *object arrays* that are either *single dimensional* or *multi dimensional*. Revising two different initialization syntaxes from Tutorial 7 on non-object arrays.
- §2.6.11 **Tutorial 11: References to another class.** When **class**es have references to objects of other classes in their **property**s then you can set up *relationships between different* **class***es*.

- §2.6.12 **Tutorial 12: Overloading methods.** *Overloading* **method***s,swapping the* **property***s* of two objects,and *converting* **method***s to* **function***s and vice-versa.*
- §2.6.13 **Tutorial 13: More about references.** More questions about references.
- §2.6.14 **Tutorial 14: Linked lists.** When a **class** has a *reference to itself* as a **property** then you can build *linked lists* out of objects of this class. **WARNING:** Linked lists are tricky for novice programmers to grasp.
- §2.6.15, **Tutorial 15: Introducing *inheritance*.** Introducing *polymorphism*, *getter* and *setter* **method**s, the **instanceof** keyword for run-time type enquiry, the `Object` class and explaining in more depth *why the* `toString` **method** *is useful for debugging.*
- §2.6.16 **Tutorial 16: Advanced inheritance.** Showing you *how inheritance can be used to reduce the amount of duplication of code.*
- §2.6.17 **Tutorial 17: Arrays, inheritance and polymorphism.** Also teaches why in most cases *it is better to use polymorphism rather than run-time type inquiry.*

2.6.1 Tutorial 1

Question 1.1: Some code to get you started. First, please visit §2.2.2 for the programs that you need to have installed before you can do any coding in J.T.W. You should then download a tarball (also known as a *compressed archive file*):

`davinpearson.com/binaries/preprocessor-YYYYmmdd-HHMMSS.tar.gz`

where `YYYY` is the year the file was last modified, `mm` is the month the file was last modified and `dd` is the day the file was modified and similarly for `HH`, `MM` and `SS`, containing the code you need to get started. Then unzip the tarball and change directory to `~/preprocessors` and issue the following command: `bash install <username>`. Note that you will need to be logged in as **root** to execute this command. If you want to run the installer module with all of the default settings, you need to execute the following command:

```
yes | bash install www
```

If you are using *M.S. Windows* and your `HOME` variable is unset, then you will need to set it to a sensible value. Examples of sensible values for your `HOME` variable include, `c:\` or `c:\home` or `d:\home` if your `d` drive is a hard drive. To set the `HOME` variable in windows, press **Windows E** and right click on *My Computer* (Windows XP) or *This Computer* (Windows 10) and click on *Properties*, then click on *Advanced system settings*, then click on *Advanced*, then click on *New* environment variable to set the `HOME` variable.

When you run the `install` script using the command `bash install <username>` and you will be prompted for the location of prefix directory and the location of the place to keep your `*.jtw` files. You will also be asked if you want to install just *Davin's* `jtw-mode` or *Davin's Full Version of GNU Emacs*. The advantage of installing Davin's Full Version of GNU Emacs is that it has been extensively modified for optimum editing of code in many different languages. To install J.T.W. using the default settings, you need to issue the following command: `yes | bash configure`, assuming you have the command `yes` installed as will be the case if you are using GNU/Linux or Cygwin[3]. Note that under the default settings, Davin's Full Version of GNU Emacs is **not** extracted.

Question 1.2: Your first J.T.W. program. Traditionally in computer science the first program that you write in any programming language is a program that does nothing else but prints out `"Hello,World"`. The following code does just that. In order to compile and run the following program you will need use the *copy* feature of your web browser and the *paste* feature of your text

[3] `www.cygwin.com`

editor (which I hope for your sake is Davin's version of GNU Emacs or GNU Emacs with Davin's `jtw-mode`) to bring the following program code out of the J.T.W. web page and into your text editor for editing purposes. Once you have copied and pasted your code you can then compile and run it. Every other question in these tutorial requires you to be familiar with the copy and paste operation unless you are a masochist and like to type in your source code by hand. In the following code, note the use of the **class** construct. In J.T.W. and Java, every piece of program code that does some real computational work resides in a **class** of some description.

```
class MyFirstProgram

   beginMain
      System.out.println(STRINGBGFG("Hello,World!"));
   endMain
```

The code for any **class** X in these tutorials should reside in a file called `X.jtw`. Therefore the above code should be put into a file called `MyFirstProgram.jtw`. If two classes X and Y use each other and X contains the **main** **function** then it is convenient to place them both in a file called `X.jtw`. To build and run some code, you first need to be in the `/jtw-tutorials` folder and secondly you need to issue the following shell command: `make build X.run` where X is the name of the **class** that you want to run, so it is

```
make build MyFirstProgram.run
```

in this case. For all questions that follow this one, it will be assumed that you know how to do this. See §2.16.6 for more information about how to build collections of classes and entire packages.

Question 1.3: Multiple calls to `System.out.println`. Change the above code from printing the string `STRINGBGFG("Hello, World!")` to printing out the following messages. Please note that it will be easiest to use multiple calls to **System.out.println**() which sends text to the screen for the purpose of viewing.

```
Hello, Anne!  How are you doing?
Hello, Brian!  How are you doing?
Hello, Clare!  How are you doing?
```

Question 1.4: Functions, parameters and arguments. A **function** is a piece of code that does some computational work and optionally returns a value. Notice how the `hello` **function** below takes a value of whose name to say hello to. This value `who` is called a *parameter*. The values passed to the parameter by the call to the **function** is called an *argument*. For the purposes of this question, add two more calls to the `hello` **function** in the **main** **function** to get the same result as the code for the previous question. The keyword *void* indicates that this **function** does not return a value. See the next question for a **function** that does return a value.

```
class MySecondProgram

   function void hello (String who)

      System.out.println(STRINGBGFG("Hello ") + who + STRINGBGFG(",how are you doing?"));

   beginMain
      hello(STRINGBGFG("Anne"));
   endMain
```

Question 1.5: Return values. Notice how the following `hello` **function** returns a string rather than printing out the string. Add two more calls to the `hello` **function** below to get the same result as for Question 1.3.

```
class MyThirdProgram

   function String hello (String who)

      return STRINGBGFG("Hello ") + who + STRINGBGFG(",how are you doing?");

   beginMain
      System.out.println(hello(STRINGBGFG("Anne")));
   endMain
```

Question 1.6: Ignoring return values. In J.T.W. and Java, it is not necessary to use a value that is returned by a **function**. Sometimes this wastes computational resources since the value that is computed by the **function** is not used but other times when the **function** whose value is to be ignored does some additional work by setting the value(s) of some variable(s) to different values then the **function** call is not a waste of resources. To ignore the value returned by the `hello` **function**, simply call the **function** without using the value like so: `hello(STRINGBGFG("Ignored"));` For the purposes of this question, try calling the `hello` **function** without using the return value by adding a line of code to the `main` **function**.

Question 1.7: Comments. Study the following code. Note the use of `dark green` and **red** comments. Comments are used to disable code for debugging purposes and also to help explain how a program works. The most useful comment in J.T.W. and Java is `/**` until the first `*/`. This type of comment is harvested by Javadoc to produce documentation on how a **class** works. The second and third most useful comments are (respectively) `//` until the end of the line and `/*` until the first `*/`. The third type of comment is not very useful because in J.T.W and Java you are not allowed to have one comment inside another, so if you use this type of comment you will constantly need to search for and remove `*/` closing comments. In the tutorials that follow you will see many comments, although mainly the first and second types of comments.

```
/** This comment is harvested by Javadoc
      to document the MyFourthProgram class */
class MyFourthProgram
begin // I am a single line comment
   /* I am
         a multi-line
               comment */
   /** This comment is harvested by Javadoc
         to document the hello function */
   function String hello (String who)
   begin
      return STRINGBGFG("Hello ") + who + STRINGBGFG(",how are you doing?");
   end
   /** This comment is harvested by Javadoc
         to document the main function */
   beginMain
      System.out.println(hello(STRINGBGFG("Anne")));
   endMain
end
```

2.6.2 Tutorial 2

Question 2.1: The following code returns whether or not the current parameter `ch` is a vowel. The parameter `ch` is of type char which is used to hold the components of a string. That is to say, strings are built out of sequences of chars. Also note the use of the `Character.toUpperCase` **function** to convert chars into uppercase chars so that the code works equally well for `isVowel(STRINGBGFG('a'))` and `isVowel(STRINGBGFG('A'))`. Study, compile and run the following code. Does it print what you expected it to? If not, then fix the bug.

```
class Scrabble

   function boolean isVowel (char ch)

      ch = Character.toUpperCase(ch);
      if ((ch == STRINGBGFG('A')) or (ch == STRINGBGFG('E')) or (ch == STRINGBGFG('I')) or
(ch == STRINGBGFG('O')) or (ch == STRINGBGFG('U')))
      then return true;
      else return false;

   beginMain
      System.out.println(isVowel(STRINGBGFG('a')));
   endMain
```

In the above code, note the difference between `a = b` example: `ch = Character.toUpperCase(ch)` and `a == b` example: `ch == STRINGBGFG('A')`. The first is an *assignment* that sets `a` to be whatever the value of `b` is, while the second is a *question* that says whether or not the two arguments `a` and `b` are equal.
Note that later on in this tutorial you will learn that this is *not* the way to compare two strings. Also note the use of the *boolean* return type. This means that the return value is either *true* or *false*.
Question 2.2: By copying the pattern established by the above code, write a **function** `isConsonant` which returns whether or not the given argument is not a vowel. The easiest way to do this is to write `isVowel(ch) == false` which means: *"ch is not a vowel"*. You will also need to ensure that the parameter `ch` is greater than or equal to `STRINGBGFG('A')` and less than or equal to `STRINGBGFG('Z')`. Then test your code by calling `isConsonant` from the **main function**.
Question 2.3: By copying the pattern established in the following code:

```
function int countVowels (String word)

   var int result = 0;
   superfor (var int i=0 to word.length()-1;)

      var char ch = word.charAt(i);
      if (isVowel(ch)) then result = result + 1;

   return result;
```

write a **function** that counts the number of consonants in a word. Note the use of the **var** keyword for defining variables that are local to functions. Local variables are very much like parameters that were introduced in the previous tutorial. In the above code, note the use of `word.charAt(i)` and `word.length()`. The first of these results *the character at location in the string* `word` *given by*

the value of `i` and the second of these returns *the length of the string* `word`. In Tutorial 11 you will learn that these are called *methods* which are different from *functions* that currently know how to write. Until we get to this tutorial and we are ready to teach you how to write your own methods, you will only call existing methods such as the above methods of the **String class**. Then test your code by calling it from the **main function**.

Question 2.4: Write a **method** `simpleScoreWord` that calls `countVowels` and `countConsonants` to give a *Simple Score* of a word. The Simple Score of a word is the number of vowels in the word plus the number of consonants in the word times ten. Then test your code by calling it from the **main function**.

Question 2.5: Write a **method** `advancedScoreLetter` that returns the *Advanced Score* of a letter. Here is a breakdown of the distribution of letters for the purpose of the calculation of the Advanced Scores.

- 2 blank tiles (scoring 0 points)
- 1 point: E 12 tiles, A 9 tiles, I 9 tiles, O 8 tiles, N 6 tiles, R 6 tiles, T 6 tiles, L 4 tiles, S 4 tiles, U 4 tiles
- 2 points: D 4 tiles, G 3 tiles
- 3 points: B 2 tiles, C 2 tiles, M 2 tiles, P 2 tiles
- 4 points: F 2 tiles, H 2 tiles, V 2 tiles, W 2 tiles, Y 2 tiles
- 5 points: K 1 tiles
- 8 points: J 1 tiles, X 1 tiles
- 10 points: Q 1 tiles, Z 1 tiles

Then test your code by calling it from the **main function**.

Question 2.6: Write a **method** `advancedScoreWord` that returns the *Advanced Score* of a word. The Advanced Score of a word is the sum of the Advanced Scores of each letter in the word. If the word is eight letters long then you should add an extra, say, 50 points to the score. Then test your code by calling it from the **main function**.

Question 2.7: Comparing strings. Amend the `advancedScoreWord` **function** so that swear words get a score of zero. For the purposes of this question you only need to think of three swear-words to add to the code. In the interests of not offending anyone, please keep your choice of swear words very tame. When comparing strings it is a mistake to use `==` which you already know is how you compare the following types that you know of so far: booleans, chars and ints. Using `==` on strings compiles and runs but gives you the incorrect result. The correct **method** to compare strings is to use the `equals` **method** of the string **class** like so: `word.equals(STRINGBGFG("bugger"))` which returns true or false, depending on whether or not the string `word` currently holds the value `STRINGBGFG("bugger")`.

Question 2.8: Change the `advancedScoreWord` **function** so it works equally well with uppercase words and lowercase words. You will need write to call either `word.toUpperCase()` or `word.toLowerCase()` and store the result in `word`.

2.6.3 Tutorial 3

Question 3.1a: For loops that count up in steps of one. Study the following code and verify that it prints out "`2 3 4 5 6 7 8 9 10`" by compiling and running it. Notice that the **System.out.print**() **function** call doesn't print a carriage return after printing the argument value. That is why the **System.out.println**() **function** call is needed at the end of the **superfor** and **for** loop, to print a carriage return at the end of the line. Also note the use of the plus sign to concatenate a string and the number to produce another string.

```
beginMain
   /* Here is the superfor loop:  */
   superfor (var int i=2 to 10) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();

   /* Here is the ordinary for loop:  */
   for (var int i=2 i<=10; i=i+1) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();
endMain
```

Question 3.1b: Change the **superfor** loop and the ordinary **for** looop to print out: "5 6 7 8 9 10".
Question 3.1c: Change the **superfor** loop and the ordinary **for** looop to print out: "234 235 236 237 238".
Question 3.1d: Change the **superfor** loop and the ordinary **for** looop to print out: the for loop to print out "48 49 50 ... 75 76".
Question 3.1e: Change the for loop to print out "-5 -4 -3 -2 -1 0 1 2 3".
Question 3.2a: For loops that count up in steps greater than one. Study the following code and verify that it prints out "10 15 20 25 30 35 40" by compiling and running it.

```
beginMain
   /* Here is the superfor loop:  */
   superfor (var int i=10 to 40 step 5) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();

   /* Here is the ordinary for loop:  */
   for (var int i = 10; i<=40; i=i+5) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();
endMain
```

Question 3.2b: Change the for loop to print out "20 25 30 35 40".
Question 3.2c: Change the for loop to print out "100 105 110 115 120 125".
Question 3.2d: Change the for loop to print out "2 4 6 8 10 12 14".
Question 3.2e: Change the for loop to print out "10 13 16 19 22 25".
Question 3.3a: For loops that count down in steps of one. Study the following code and verify that it prints out "10 9 8 7 6 5 4 3 2 1" by compiling and running it.

```
beginMain
   /* Here is the superfor loop:  */
   superfor (var int i=10 downto 1) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();

   /* Here is the ordinary for loop:  */
   for (var int i = 10; i>=1; i=i-1) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();
endMain
```

Question 3.3b: Change the for loop to print out "10 9 8 7 6 5 4".
Question 3.3c: Change the for loop to print out "20 19 18 17 16 15 14 13 12".
Question 3.3d: Change the for loop to print out "66 65 64 ... 47".
Question 3.3e: Change the for loop to print out "3 2 1 -1 -2 -3 -4 -5 -6 -7".
Question 3.4a: For loops that count down in steps greater than one. Study the following code and verify that it prints out "100 90 80 70 60 50 40 30 20" by compiling and running it.

```
beginMain
   /* Here is the superfor loop:  */
   superfor (var int i=100 downto 20 step -10) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();

   /* Here is the ordinary for loop:  */
   for (var int i = 100; i>=20; i=i-10) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();
endMain
```

Question 3.4b: Change the for loop to print out "80 70 60 50 40 30 20".
Question 3.4c: Change the for loop to print out "500 490 480 470 460".
Question 3.4d: Change the for loop to print out "10 8 6 4 2 0".
Question 3.4e: Change the for loop to print out "33 28 23 18 13 8 3".
Question 3.5a: For loops that use floating point numbers to count. Study the following code and verify that it prints out "1.1 2.2 3.3 4.4" by compiling and running it. The type name *double* is short for *double precision floating point*. It is natural to ask: why not use single precision floating point? The answer to this question is that double precision floating point gives fewer compilation errors than single precision floating point does.

```
beginMain
   /* Here is the superfor loop:  */
   superfor (var double i=1.1 to 4.41 step 1.1) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();

   /* Here is the ordinary for loop:  */
   for (var double i = 1.1; i<=4.41; i=i-1.1) System.out.print(STRINGBGFG(" ") + i);
   System.out.println();
endMain
```

Note the extension of the **to** part of the superfor loop and the second part of the **for** loop. The number is 4.41 and this prevents round off errors in doubles from getting to the final value of 4.4.
Question 3.5b: Change the for loop to print out "0 2.2 4.4 6.6". Note that rounding errors may prevent you from getting this exact answer. Also note that the answer to this question is not what you would naively expect without running the code.
Question 3.5c: Change the for loop to print out "-30 -19.9 -9.8 0.3 10.4 20.5".
Question 3.5d: Change the for loop to print out "100.0 96.7 93.4 90.1 86.8 83.5 80.2 76.9".
Question 3.5e: Change the for loop to print out "-100.0 -105.5 -111.0 -116.5".
Question 3.6a: For loops that use chars to count. Study the following code and verify that it prints out "a b c d e f g h i j k l m n o p q r s t u v w x y z" by and running it.

```
beginMain
   /* Here is the superfor loop:  */
   superfor (var char i = STRINGBGFG('a') to STRINGBGFG('z'))
   System.out.println();

   /* Here is the ordinary for loop:  */
   for (var char i=STRINGBGFG('a'); i<=STRINGBGFG('z'); i=i+1) System.out.print(STRINGBGFG("
") + i);
   System.out.println();
endMain
```

Question 3.6b: Change the for loop to print out "`a b c d e f`".
Question 3.6c: Change the for loop to print out "`z y x w v u t s r q p o n m l k j i h g f e d c b a`".
Question 3.6d: Change the for loop to print out "`p o n m l k j i h`".
Question 3.6e: Change the for loop to print out "`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`".

2.6.4 Tutorial 4

Study the following code:

```
class LoopTest

   function int powerOf2A (int n)

      var int counter = n;
      var int result = 1;
      while (counter != 0)

         result = 2 * result;
         counter = counter - 1;

      return result;


   function int powerOf2B (int n)

      var int counter = n;
      var int result = 1;
      do

         result = 2 * result;
         counter = counter - 1;
      while (counter != 0);
      return result;


   function int powerOf2C (int n)

      var int result = 1;
      for (var int counter = n; counter != 0; counter = counter - 1)

         result = 2 * result;

      return result;


   function int powerOf2D (int n)

      var int result = 1;
      superfor (var int counter = n downto 1)

         result = 2 * result;
```

```
    return result;



/**
* Prints a row of stars of a given length.
*/
function void printLineC (int length)

    for (var int i = 0; i<length; i=i+1)

        System.out.print(STRINGBGFG("#"));

    System.out.println();


beginMain
    // For question 4.1 add some code here...
endMain
```

Question 4.2: To the **main function** add some code to call the functions `powerOf2A`, `powerOf2B`, `powerOf2C` and `powerOf2D` to verify that they all return the same result. To inspect the result you will need to apply the **System.out.println**() statement to the values returned by those functions.
Question 4.3: There is a bug in the `powerOf2B` **method** because it does not behave correctly in the case when n is zero. Put an *if* statement at the top of this **method** to make it handle the case of zero properly.
Question 4.4: By copying the pattern of `powerOf2A`, `powerOf2B`, `powerOf2C` and `powerOf2D`, write methods `printLineA`, `printLineB` and `printLineD` that work identically to the **method** `printLineC`, except that they use *while* loops, *do* loops and **superfor** loops, respectively. Add some code to the **main function** to test them out.
Question 4.5: Based on the previous three questions, is there a best looping construct? Or does it depend on what the looping construct is going to be used for?

2.6.5 Tutorial 5

Question 5.1: Study the following code and then compile and run it to verify that it prints out the lyrics to a popular beer-drinking song:

```
class BeerSong

beginMain
    System.out.println(STRINGBGFG("Five bottles of beer on the wall."));
    System.out.println(STRINGBGFG("Five bottles of beer on the wall."));
    System.out.println(STRINGBGFG("If one bottle of beer should accidentally fall,"));
    System.out.println(STRINGBGFG("there'd be four bottles of beer on the wall."));
    System.out.println();
    System.out.println(STRINGBGFG("Four bottles of beer on the wall."));
    System.out.println(STRINGBGFG("Four bottles of beer on the wall."));
    System.out.println(STRINGBGFG("If one bottle of beer should accidentally fall,"));
    System.out.println(STRINGBGFG("there'd be three bottles of beer on the wall."));
    System.out.println();
    System.out.println(STRINGBGFG("Three bottles of beer on the wall."));
```

```
    System.out.println(STRINGBGFG("Three bottles of beer on the wall."));
    System.out.println(STRINGBGFG("If one bottle of beer should accidentally fall,"));
    System.out.println(STRINGBGFG("there'd be two bottles of beer on the wall."));
    System.out.println();
    System.out.println(STRINGBGFG("Two bottles of beer on the wall."));
    System.out.println(STRINGBGFG("Two bottles of beer on the wall."));
    System.out.println(STRINGBGFG("If one bottle of beer should accidentally fall,"));
    System.out.println(STRINGBGFG("There'd be one bottle of beer on the wall."));
    System.out.println();
    System.out.println(STRINGBGFG("One bottle of beer on the wall."));
    System.out.println(STRINGBGFG("One bottle of beer on the wall."));
    System.out.println(STRINGBGFG("If one bottle of beer should accidentally fall,"));
    System.out.println(STRINGBGFG("there'd be no bottles of beer on the wall."));
    System.out.println();
endMain
```

Question 5.2: The following is the first attempt to make the code smaller but to keep the same output: If you compile and run the following code you will notice that it counts up from one rather than down from `n`. Change the *for* loop so that it runs down rather than up. For information about how to write the for loop, please consult Tutorial 2.

```
class BeerSong

  function song (int n)

    for (var int i=1; i<=n; i=i+1)

      System.out.println(i + STRINGBGFG(" bottles of beer on the wall"));
      System.out.println(i + STRINGBGFG(" bottles of beer on the wall"));
      System.out.println(STRINGBGFG("If one bottle of beer should accidentally fall,"));
      System.out.println(STRINGBGFG("there'd be ") + (i-1) + STRINGBGFG(" bottles of beer
on the wall"));
      System.out.println();



  beginMain
    song(5);
  endMain
```

Question 5.3: Finish the `number2string` **function** below and add a new **function** call to this **function** in the song **function** so that it print textual numbers rather than digits.

```
function String number2string (int n)

  assert n>=0 :  n;
  assert n<=10:  n;
  if (n == 0) then return STRINGBGFG("no");
  if (n == 1) then return STRINGBGFG("one");
  if (n == 2) then return STRINGBGFG("two");
  /* rest of code goes here */
```

```
if (n == 9) then return STRINGBGFG("nine");
if (n == 10) then return STRINGBGFG("ten");
assert false;
```

Question 5.4: Add a new **function** String **capitalize** (int n) that capitalizes the first word in a String and call this **function** from the song **function** so that the first words in each sentence are capitalized. You should find the **function** Character.toUpperCase and the methods String and String helpful for writing this **function**. See the String **class** of the java.lang package in the following link:

docs.oracle.com/javase/1.5.0/docs/api

for more details.

Question 5.5: Add new **function** call String **plural** (int n) that returns the string STRINGBGFG("s") if n is not equal to 1 and the empty string STRINGBGFG("") otherwise. Then call this **function** from the song **function** so that the phrase STRINGBGFG("bottle") is pluralized when it should be.

Question 5.6: Write a **function** called number2string2 that can handle values up to but not including 100. Note that you will need multiple **if** statements to achive this. Note that if n is a number then the following expressions are useful:

- **var** int temp1 = n / 10 % 10 results in temp1 holding the tens digit of n and is zero in the case that n<10.
- **var** int temp2 = n % 10 results in temp2 holding the ones digit of n.

Also make it print out STRINGBGFG("one hundred or more") in the case that n>=100

Question 5.7: Change the song **function** so that the following **function** call: song(5,STRINGBGFG("rum")); in the **main** **function** results in the following printout:

```
Five bottles of rum on the wall.

...

there'd be no bottles of rum on the wall.
```

Question 5.8: Once all the code is working, add the following line to the **main** **function**: song(100,STRINGBGFG("gin")); so that it prints out the following:

```
One hundred bottles of gin on the wall.

...

there'd be zero bottles of gin on the wall.
```

Question 5.9 Write a new **function** number2string3 that works like number2string2 and number2string except that it handles numbers up to 999. Internally number2string3 should call number2string2. You might find the following **function** useful:

```
function String textand (String a, String b)

  if (a.equals(STRINGBGFG("")) or b.equals(STRINGBGFG(""))) then return a + b;
  else return a + STRINGBGFG(" and ") + b;
```

Question 5.10 † **Tricky** Write a new **function** `number2string4` that works like `number2string3` execpt that it handles numbers up to nine hundred and ninety-nine million nine hundred and ninety-nine thousand nine hundred and ninety-nine, i.e. 999,999,999. The **function** `number2string4` should internally call `number2string3` like so:

- `var String ones = number2string3(n % 1000);`
- `var String thousands = number2string3(n / 1000 % 1000);`
- `var String millions = number2string3(n / 1000 / 1000 % 1000);`

Note that the variables above will have values from 0 to 999 inclusive.

2.6.6 Tutorial 6

Question 6.1: Study, compile and run the following code. Note the use of the *class variable* `myMoney`. A **class** variable is different from a variable that is local to a **function** because the lifetime of the **class** variable is for the duration that the program is run, whereas the lifetime of a local variable is for the duration of the **function** call. In the code that follows, the variable `myMoney` is used to store a numerical value, for how much money you have.

```
class Money

   /** Property myMoney stores money value in dollars */
   classVar int myMoney;

   function void spend (String item, int value)

      myMoney = myMoney - value;
      System.out.println(STRINGBGFG("*** spent $") +
                         value +
                         STRINGBGFG(" on ") + item +
                         STRINGBGFG(",leaving you with $") + myMoney);


   beginMain
      myMoney = 100;
      spend(STRINGBGFG("aquarium"),50);
      spend(STRINGBGFG("shoes"),100);
      spend(STRINGBGFG("lipstick"),20);
   endMain
```

Question 6.2: Change the `myMoney` **class** variable so that it is a *double* (short for double-precision floating point) rather than an *int*. You will need to add a new **function** `money2string` that converts double values into strings. For example the floating point number 1.2345 should be printed out as $1.23. If `x` is a double then the following expression converts `x` from a double into a number of dollars `(int)x` and the following expression converts `x` into a number of cents `(int)(money * 100) - 100 * dollars`. Note that you will need to make it so that $1.03 prints out as this value.

Question 6.3: Add an *if* statement to the `spend` **function** so that it uses **System.out.println**() to print out an error message if the person does not have enough funds in their bank account to pay for the item parameter.

Question 6.4: Add a new **class** variable `double governmentsMoney` and make it so that 12.5goes to the government in the form of G.S.T. (¡u¿G¡/u¿oods and ¡u¿S¡/u¿ervices ¡u¿T¡/u¿ax a value-added tax)
Question 6.5: Add a new **class** variable `numBattleships` that records how many batteships are owned by the government. Write a **function** `buyBattleShips` that causes the government to buy as many battleships as it can afford. Make it so that the `buyBattleShips` **function** prints out how many battleships were purchased. Let the cost of each battleship be one million dollars and store this value in a variable called `costOfShip`. Please note that if the government's money is less the one million dollars then no battleships will be purchased.
Question 6.6: Set the initial value for `governmentsMoney` to be two millions dollars, then call the `buyBattleShips` **function** and verify that two battleships were purchased.

2.6.7 Tutorial 7

This tutorial teaches you how to create single dimensional and multi-dimensional arrays of non-objects. The non-object types in Java are those which aren't declared inside a **class**, so it includes the following types: *boolean*, *char*, *int*, *float* and *double*. A helpful convention in Java is that the non-object types start with a lowercase letter, while object types start with an uppercase letter, such as for example the `String` **class** as an example of an object type. In addition to this, two different array initialization syntaxes are presented.

Single dimensional arrays

Question 7.1: Here is an example of a convenient one dimensional array initialization syntax. Study, compile and run the following code. The code `int[]` should be read out loud as *int array* indicating that the variable `a` is an int array, also known as an array of ints. Note that the first value of the for loop below is zero. This is because in J.T.W. and Java, the first index of an array is zero not one. This convention harks back to the old days of the *C Programming Language* and is used because it is more efficient in the low level of machine language than counting arrays from one. Also note that parenthesis are used to delimit arrays. I use this practice because this is the only place in Java where a semicolon follows a closing parenthesis. If you don't know what I am talking about, simply ignore that remark!

```
var int[] a = { 1,2,3 };
for (var int i=0; i<3; i=i+1)

   System.out.println(STRINGBGFG("a[") + i + STRINGBGFG("]=") + a[i]);
```

Due to a design oversight by the creators of Java you cannot use this syntax to re-initialize an array like so:

```
a = { 4,5,6 }; // Compilation error
```

Luckily there is a way array around this oversight and that is to use a design pattern where you introduce a temporary variable like so:

```
var int[] temp = { 4,5,6 };
a = temp; // Array "a" now holds 4 5 6
```

Later you will learn why this design pattern is useful for re-initializing multi-dimensional arrays.
Question 7.2: Write a **function** `print` that takes an int array argument and prints out the array. You will need to use the `length` property of the array parameter so your **function** works

with arbitrary sized arrays. Change the **main** **function** to what follows so that it contains a call to the `print` **function**.

```
var int[] a = { 1,2,3 };
print(a);
```

Question 7.3: Write a **function** with same name as the previous `print` **function**, except that this one should take an argument that is a `double[]`, also known as a double array. Two functions with the same name in the same **class** is allowed in Java and the practice of using has a special name that is: **function** *name overloading*. Overloading is only allowed when the two functions with the same name have different parameters. When you call an overloaded **function** J.T.W. and Java looks at the number and types of the arguments a determines from this which of the overloaded functions to call. Change the **main** **function** to what follows so that it initializes an array of double-precision floating point variables and then calls the second `print` **function**.

```
var double[] b = { 1.1,2.2,3.3 };
print(b);
```

Here is an example of a second initialisation syntax. For this particular example it is better to use the simpler, earlier initialisation syntax, but when the size of the array to be created is to be determined at run-time, then the second syntax should used. The next question will show you an example of this.

```
beginMain
   var int[] a = new int[3];
   // at this point the array is all zeroes
   for (var int i=0; i<3; i=i+1)

      a[i] = i;

   print(a);
endMain
```

Question 7.4: Write a **function** `create` takes one int argument, the size of the array to create and returns an int array of that size. Make it so the $\texttt{i}^{th}$ element of the array is initialized to `i`. Call this **function** from the **main** **function** like so:

```
beginMain
   var int[] a = create(3);
   print(a);
endMain
```

Question 7.5: Write a **function** `create2` takes one int argument, the size of the array to create and returns a double array of that size. Make it so the $\texttt{i}^{th}$ element of the array is initialized to `i.i`, given that i¡10. Why is it not possible to overload that `create` **function**? Try it and see what the compiler says. Call `create2` from the **main** **function** like so:

```
beginMain
   var double[] a = create2(3);
   print(a);
endMain
```

Question 7.6: Write a **function** `doubler` that takes an int array `x` and returns a new int array `result` that is twice as big as `x`. Copy `x` into `result` before you return it. The extra elements in the `result` should all be zero.
Question 7.7: Change the `doubler` **function** so that every zero in the array `result` is set to the value 13.

Two dimensional arrays

Question 7.8: Here is an example of a convenient two dimensional array initialization syntax. Study, compile and run the following code. The code `int[][]` should be read out loud as *int array array* indicating the variable `a` is an *int array array*, also known as a *two-dimensional array of ints*.

```
beginMain
   var int[][] a = { { 1,2,3 } { 4,5 } { 6 } }

   for (var int y=0; y<a.length; y=y+1)

      for (var int x=0; x<a[y].length; x=x+1)

         System.out.print(STRINGBGFG(" ") + a[y][x]);

      System.out.println();

endMain
```

Question 7.9: By copying the pattern of the code above, do some more overloading of the `print` **function** by writing two new `print` functions, one taking a two-dimensional array of ints, the other taken a two-dimensional array of doubles. The call both of these functions from the **main function**.
Note that if `x` is a two-dimensional array of ints, then `x[i]` is a one dimensional array of ints for each in the range `0 ... x.length-1`. Note that in the above code, `a[0]` is an array of three ints, `a[1]` is an array of two ints and `a[2]` is an array of one int. The reason these sub-arrays are all of different sizes is to save your computer's precious memory. For example you can have one sub-array much longer than all of the others without needing to allocate a whole bunch of memory that will go unused. Since `a[0]` is an int array, you would naively expect it to be able to be re-initialized like so:

```
a[0] = { 4,5,6,7};
```

so that after this code `a[0]` holds the four element long array 4,5,6 and 7. But as mentioned above in Section §7.1, this doesn't work because of a design oversight by the creators of Java. Luckily as mentioned above there is a way around this oversight and that is to use a temporary variable like so:

```
var int[] temp = { 4,5,6,7};
a[0] = temp; // Array "a[0]" now holds 4 5 6 7
```

Like with one dimensional arrays, there is a second initialisation syntax for two-dimensional arrays and here it is. Unlike the above code the sub-arrays `a[0]`, `a[1]` and `a[2]` are all of equal size, namely three.

```
var int[][] a = new int[3][3];
a[0][0] = 1; a[1][0] = 2; a[2][0] = 3;
a[0][1] = 4; a[1][1] = 5;
a[0][2] = 6;
```

Question 7.10: Write a **function** `create3` and `create4` that takes on int argument `size` and returns a two dimensional array of ints or doubles, respectively. Make is so that if `a` is the name of the returned array, then `a[y][x]` is set to the value of `x+y`.

Three dimensional arrays

Question 7.11: Using the knowledge you have gained so far about arrays, create, initialize and print a three dimensional array of ints.

2.6.8 Tutorial 8

Question 8.1: Study, compile and run the following code which resides in a file called `Box.jtw`. Notice the use of **System.out.print**() to print without a trailing newline and **System.out.println**() to print with a trailing newline. The **ln** part tells you this.

```
class Box

   function void square (int n)

      for (var int y=0; y<n; y=y+1)

         for (var int x=0; x<n; x=x+1)

            if ((x == 0) or (x == n-1) or (y == 0) or (y == n-1))
            then System.out.print(STRINGBGFG("#"));
            else System.out.print(STRINGBGFG(" "));

         System.out.println();


   beginMain
      square(5);
   endMain
```

Notice that here is the output of the above code for different values of the `n` parameter:

n = 1	#
n = 2	## ##
n = 3	### # # ###
n = 4	#### # # # # ####
n = 5	##### # # # # # # #####

Question 8.2: By copying the pattern established in the above code, write a new **function** `square2` that generates the following output. Note that you will need to remove some of the *or* clauses in the `square` **method** above to get the following output:

n = 1	#
n = 2	## ##
n = 3	### ###
n = 4	#### ####
n = 5	##### #####

Question 8.3: By copying the pattern established in the above code, write a now **function** `square3` that generates the following output:

n = 1	#
n = 2	## ##
n = 3	# # # # # #
n = 4	# # # # # # # #
n = 5	# # # # # # # # # #

Question 8.4: Study, compile and run the following code which resides in a file called `Box.java`:

```
class Box

   function void x (int n)

      for (var int y=0; y<n; y=y+1)

         for (var int x=0; x<n; x=x+1)

            if ((x == y) or (x == n-1-y)) then System.out.print(STRINGBGFG("#"));
            else System.out.print(STRINGBGFG(" "));

         System.out.println();


   beginMain
      x(5);
```

Notice that here is the output of the above code for different values of the `n` parameter:

n = 1	#
n = 2	## ##
n = 3	# # # # #
n = 4	# # ## ## # #
n = 5	# # # # # # # # #

Question 8.5: By copying the pattern established in the above code, write a now **function** `x2` that generates the following output. Note that you will need to remove one of the *or* clauses in the **x method** above to get the following output:

n = 1	#
n = 2	# #

n = 3	#
	  #
	  #

n = 4	#
	  #
	  #
	  #

n = 5	#
	  #
	  #
	  #
	  #

Question 8.6: By copying the pattern established in the above code, write a now **function** `x3` that generates the following output. Note that you will need to remove one of the *or* clauses in the `x` **method** above to get the following output:

n = 1	#
n = 2	# #
n = 3	# # #
n = 4	# # # #

n = 5	#
	   #
	  #
	 #
	#

Question 8.7: Study, compile and run the following code which resides in a file called `Box.java`:

```
class Box

  function void triangle (int n)

    for (var int y=0; y<n; y=y+1)

      for (var int x=0; x<n; x=x+1)

        if (x<y)
        then System.out.print(STRINGBGFG("#"));
        else System.out.print(STRINGBGFG(" "));

      System.out.println();


beginMain
  triangle(5);
```

endMain

Notice that here is the output of the above code for different values of the `n` parameter:

n = 1	#
n = 2	# ##
n = 3	# ## ###
n = 4	# ## ### ####
n = 5	# ## ### #### #####

Question 8.8: By copying the pattern established in the above code, write a now **function** `triangle2` that generates the following output. Note that you will need to change the if clause in the `triangle` **method** above to get the following output:

n = 1	#
n = 2	## #
n = 3	### ## #
n = 4	#### ### ## #
n = 5	##### #### ### ## #

Question 8.9: Write a now **function** called `box` that generates the following output. Note that you will need to modify the `triangle` **method** above to get the following output:

n = 1	#
n = 2	## ##
n = 3	### ### ###
n = 4	#### #### #### ####
n = 5	##### ##### ##### ##### #####

Question 8.10: Add the following code to `Box.java`:

```
class Grid

   /** The dimensions of the array named: array. */
   classVar int size = 20;

   /* NOTE: the array below is a two-dimensional array */
   classVar boolean[][] array = new boolean[SIZE][SIZE];

   function void set (int x, int y, boolean v)

      if (x>=0 and x<size and y>=0 and y<size) then

         array[x][y] = v;



   function void print (int size)

      for (var int y=0; y<size; y=y+1)

         for (var int x=0; x<size; x=x+1)

            if (array[x][y])
            then System.out.print(STRINGBGFG("#"));
            else System.out.print(STRINGBGFG(" "));

         System.out.println();

      System.out.println(); // prints an empty line between shapes
```

Question 8.11: The following question will guide you through the process of making the drawing algorithm more powerful. Instead of printing the shapes directly to the screen, they will be stored in an array to be printed out only when the array has been completely set. You don't need to know a great deal about arrays to answer the remaining questions of this section as the array

code has been written for you in the Grid **class** above. For every call to **System.out.println**() in `Box.java`, replace it with a call to the set **method** of the Grid **class**. Note that the third parameter in the `set` **method** is of type *boolean*, that is to say it can be either `true` or `false`. To call a **function** of another **class** you need to prefix the name of the **class** like so: `Grid.set(/* argument values */)`. Finally at the end of all of the functions in the Box **class** except for the **main** **function** you will need to call the `print(int)` **method** of the Grid **class** to actually print out the array.

Question 8.12: Re-initialize the boolean array array named `array` from the **main** **function** of the `Box` **class**. **HINT:** to access a **class** variable from another **class**, you need to prefix it with the name of its **class** name, in this case it is `Grid`. Re-initialize the `array` variable to a two-dimensional array of dimensions 100 x 100. Also set the `size` variable to 100 so that the functions of the `Grid` **class** still work.

2.6.9 Tutorial 9

Elementary classes: using a single class for everything

For the purpose of the text that follows, O.O.P. stands for *¡u¿Oj/u¿bject ¡u¿Oj/u¿riented ¡u¿Pj/u¿rogramming*.

Question 9.1: Study, compile and run the following code:

```
// BEGIN FILE: jtw-tutorials/Person-1.jtw
class PersonDriver1
begin

   classVar String homersName = "Homer Simpson";
   classVar int    homersAge  =  40; // Homer's age in years

   classVar String fredsName  = "Fred Flintstone";
   classVar int    fredsAge   =  45; // Fred's age in years

   classVar String darthsName = "Darth Vader";
   classVar int    darthsAge  =  55; // Darth's age in years

   function void growHomer ()
   begin
      homersAge = homersAge + 1;
   end
   function void growFred ()
   begin
      fredsAge = fredsAge + 1;
   end
   function void growDarth ()
   begin
      darthsAge = darthsAge + 1;
   end

   function void knightHomer ()
   begin
      homersName = "Sir " + homersName;
   end
   function void knightFred ()
   begin
      fredsName = "Sir " + fredsName;
   end
   function void knightDarth ()
   begin
      darthsName = "Sir " + darthsName;
   end

   function void printHomer ()
   begin
```

```
      System.out.println "I am " + homersName + ", my age is " + homersAge;
   end
   function void printFred ()
   begin
      System.out.println "I am " + fredsName + ", my age is " + fredsAge);
   end
   function void printDarth ()
   begin
      System.out.println "I am " + darthsName + ", my age is " + darthsAge);
   end

   beginMain
      growHomer();
      knightHomer();
      printHomer();
      printFred();
      printDarth();
   endMain
end
// END FILE:    jtw-tutorials/Person-1.jtw
```

)

Question 9.2: By copying the pattern established in the existing code write a some new **class** variables to represent a new person called Barack Obama. Note that he was born in 1945 so at the time of writing this manual he is 67 years old.
Question 9.3: Then write some functions to work with this new person.
Question 9.4: Finally call those functions from the **main function**.

Improved classes: one object per class

As your program gets large (say over 1000 lines) then it becomes no longer practical to put all of your code in the same **class**. So it is natural to put each piece of related code in its own **class**.
Question 9.5: Study, compile and run the following code: Each of these classes can be put in their own file. For each **class** X, this **class** can be put into a file called `X.jtw`. However for the purposes of this tutorial you will probably find it easier to merge all of the classes into the same file into a file called `PersonDriver2.jtw`

```
// BEGIN FILE: jtw-tutorials/Person-2.jtw
class Homer
begin
   classVar String name = "Homer Simpson";
   classVar int    age  = 40; // Homer's age in years

   function void grow ()
   begin
      age = age + 1;
   end
   function void knight ()
   begin
      name = "Sir " + name;
   end
   function void print ()
   begin
      System.out.println "I am " + name + ", my age is " + age);
   end
end

class Fred
begin
   classVar String name = "Fred Flintstone";
   classVar int    age  = 45; // Fred's age in years
```

```
   function void grow ()
   begin
      age = age + 1;
   end
   function void knight ()
   begin
      name = "Sir " + name;
   end
   function void print ()
   begin
      System.out.println "I am " + name + ", my age is " + age);
   end
end

class Darth
begin
   classVar String name = "Darth Vader";
   classVar int    age  = 55; // Darth's age in years

   function void grow ()
   begin
      age = age + 1;
   end
   function void knight ()
   begin
      name = "Sir " + name;
   end
   function void print ()
   begin
      System.out.println "I am " + name + ", my age is " + age);
   end
end

class PersonDriver2
begin
   beginMain
      Homer.grow();
      Fred.knight();
      Homer.print();
      Fred.print();
      Darth.print();
   endMain
end
// END FILE:    jtw-tutorials/Person-2.jtw
```

Question 9.6: By copying the pattern established in the existing code write a new **class** to represent Barack Obama.
Question 9.7: Call the functions from the **main** **function** of the driver **class**.

True O.O.P.: more than one object per class

To allow for more than one object per **class**, most if not all **class** variables needs to be made into what are called *instance variables* (or more simply and more commonly known as *properties*) and most if not all functions need to be made into what are called *methods*.
Question 9.8: Study, compile and run the following code:

```
// BEGIN FILE: jtw-tutorials/Person-3.jtw
class Person
begin
   //
```

```
// NOTE: the use of the "property" keyword here instead of the "classVar" keyword
//
property String name;
property int    age; // Person's age in years

//
// NOTE: the use of the "method" keyword here instead of the "function" keyword
//
method void grow()
begin
   age = age + 1;
end

method void knight()
begin
   name = "Sir " + name;
end

method void print()
begin
   System.out.println("I am " + name + ", my age is " + age);
end

beginMain

   var Person h = new Person();
   h.name   = "Homer Simpson";
   h.age    = 40;

   var Person f = new Person();
   f.name   = "Fred Flintstone";
   f.age    = 45;

   var Person d = new Person();
   d.name   = "Darth Vader";
   d.age    = 55;

   h.grow();
   h.knight();
   h.print();
   f.print();
   d.print();

endMain
end
// END FILE:    jtw-tutorials/Person-3.jtw
```

In the above code, note the use of three references `h`, `f` and

Question 9.9: By copying the pattern established in the existing code add some code to the **main** **function** add some code to create a new person for Barack Obama.

A common design pattern: private properties, public constructor and public getters

A common design pattern in Java and one that I present for you in the following code is to make all of the properties of a **class** effectively read-only to all client classes by making all of the properties *private* and providing *non-private getter* methods for getting the values of the properties. It is possible for the original **class** to change the values of the properties but other classes (such as PersonTest below) are not capable of doing this, without calling a **method** of the original **class** such the `grow` and `knight` methods of the Person **class**. Finally an additional thing known as a **constructor** is used to ensure that objects are initialized with meaningful values for their properties.

Question 9.10: Study, compile and run the following code:

```
// BEGIN FILE: jtw-tutorials/Person-4.jtw
class Person
begin

   private property String name;
   private property int    age; // Age in years

   //
   // NOTE: Getter methods
   //
   method String getName()
   begin
      return name;
   end

   method int getAge()
   begin
      return age;
   end

   constructor(String aName, int anAge)
   begin
      this.name = aName;
      this.age  = anAge;
   end

   method void grow()
   begin
      age = age + 1;
   end

   method void knight()
   begin
      name = "Sir " + name;
   end

   method void print()
   begin
      System.out.println("I am " + name + ", my age is " + age);
   end
end

class PersonDriver3
begin
   beginMain

      //
      // NOTE: In the following constructor calls the age and name are set by the constructor
      //
      var Person h = new Person("Homer Simpson",40);
      var Person f = new Person("Fred Flintstone",45);
      var Person d = new Person("Darth Vader",55);

      h.grow();
      h.knight();
      h.print();
      f.print();
      d.print();

      h.name = "Luke Skywalker";                // ERROR: name is private
      h.age = h.age + 1;                        // ERROR: age is private

      System.out.println("name=" + h.name);     // ERROR: name is private
```

```
        System.out.println "age=" + h.age);        // ERROR: age is private

        System.out.println "name=" + h.getName(); // OK: getter is non-private
        System.out.println "age=" + h.getAge();   // OK: getter is non-private

    endMain

end
// END FILE:    jtw-tutorials/Person-4.jtw
```

Question 9.11: By copying the pattern established in the existing code add some code to the **main function** add some code to create a new person called Barack Obama.

Comparing strings

Question 9.12: Add a **method** `unknight()` which removes the `STRINGBGFG("Sir ")` title if he has one. One trap for young players in J.T.W. or Java is to use the operator `==` to compare strings like so:

```
function boolean myCompare(String a, String b)
begin
    return a == b; // Works but not as expected!
end
```

It compiles without error, but doesn't give you the result you were expecting. Instead you need to use the `equals` **method** of the String **class** like so:

```
function boolean myCompare(String a, String b)
begin
    return a.equals(b);
end
```

More generally, if `x` and `y` are a references to objects, then `x == y` returns whether or not x and y are pointing to the same object, whereas `x.equals(y)` returns whether or not the *contents* of the objects referred to by `x` and `y` are equal. The meaning of the word *contents* varies from **class** to **class**, but in the case of strings it means that the strings contain the same data.

You will also find the String **class**' `substring` and (`toUpperCase` or `toLowerCase`) methods useful here too. See the String **class** of the `java.lang` package in the following link:

docs.oracle.com/javase/1.5.0/docs/api

for more details of these two methods.

The `null` value for references

As soon as you learn how to use references you need to know that all reference variables could conceivably hold the value *null*, meaning *no value*. In particular when properties are themselves references as you will discover in Tutorial 11, then those properties are initialized to `null` by default. Object arrays that you will learn about in Tutorial 10 using the second of two initialization syntaxes are also initialized to `null` by default.

Why the `toString` method is better than any other method or property for debugging

If `x` is a reference to a **class** X (including `this`) and if `m` is a **method** of X and `p` is a property of X, and if `x` is currently `null`, then the following lines result in a `NullPointerException` being thrown when executed:

```
x.p;
x.m();
```

whereas if `x` is `null` then

- **System.out.println**`(x);` and
- **System.out.println**`(STRINGBGFG("x=") + x);`

prints out, respectively:

- `null`, and
- `x=null`.

If `x` is not `null`, it calls

- **System.out.println**`(x.toString());`
- **System.out.println**`(STRINGBGFG("x=") + x.toString());`

so these expressions are safer to use than any other **method** or property in situations where `x` might be `null`. The syntax of the `toString` **method** is as follows:

```
public method String toString()

   // Code goes here...
```

Importantly for reasons which will be explained later the `toString` **method** must be declared with `public` visibility. For other properties and methods to be used safely with `null` references you need to wrap a conditional *if* construct around the calling of the **method** or property like so for properties:

```
if (x != null)
then
   System.out.println(x.p);
```

or like so for methods:

```
if (x != null)
then
   System.out.println(x.m());
```

Therefore the `toString` **method** is more convenient than any other **method** or property. Note that its use is without the explicit call to the `toString` **method** and only used with a variable name, including `this` for the current **class**. Most of the time the `this` keyword is optional which is why novices don't bother to learn it, but in the case of the `toString` **method** it is essential, as can be seen in the following example code:

```
System.out.println(STRINGBGFG("x.toString()=") + x);
System.out.println(STRINGBGFG("this.toString()=") + this);
```

Question 9.13: Change the `print` **method** above from a **method** that prints out to the screen to a **method** called `toString` that returns a string.
Question 9.14: Call the `toString` **method** instead of the `print` methods in the **main** **function**.

2.6.10 Tutorial 10

This tutorial teaches you how to create single dimensional and multi-dimensional arrays of objects. The object types are all types execpt for *boolean*, *char*, *int*, *float* and *double*. A helpful convention in Java is that the Object types start with an uppercase letter, while non-object types start with a lowercase letter, such as for example the **String** **class** as an example of an object type. In addition to this, two different array initialization syntaxes are presented.

Single dimensional arrays

Question 10.1: Here is an example of a convenient one dimensional array initialization syntax. Study, compile and run the following code. The code `Person[]` should be read out loud as *person array* indicating the variable `a` is a *person array*, also known as an *array of persons*.

```
class Person

   private property String name;

   public constructor Person(String aName)

      name = aName;


   public String toString()

      return name;



class PersonTest

   beginMain
      var Person[] a = { new Person(STRINGBGFG("P # 1")), new Person(STRINGBGFG("P #
2")), new Person(STRINGBGFG("P # 3")) };

      for (var int i=0; i<3; i=i+1)

         System.out.println(STRINGBGFG("a[") + i + STRINGBGFG("]=") + a[i]);

   endMain
```

Due to a design oversight by the creators of Java you cannot use this syntax to re-initialize an array like so:

```
// Compilation error
a = { new Person(STRINGBGFG("P # 4")), new Person(STRINGBGFG("P # 5")), new Person(STRINGBGFG("P
# 6")), new Person(STRINGBGFG("P # 7")) };
```

Luckily there is a way array around this oversight and that is to use a design pattern where you introduce a temporary variable like so:

```
// No error
var Person[] temp = { new Person(STRINGBGFG("P # 4")), new Person(STRINGBGFG("P # 5")), new Per-
son(STRINGBGFG("P # 6")), new Person(STRINGBGFG("P # 7")) };
a = temp; // Array "a" now holds P # 4,P # 5,P # 6,P # 7
```

Later you will learn why this design pattern is useful for re-initialising multi-dimensional arrays.
Question 10.2: Write a **function** in the **class** PersonTest called `print` that takes a Person array argument and prints out the array. You will need to use the `length` property of the array parameter so your **function** works with arbitrary sized arrays. Change the **main** **function** to what follows so that it contains a call to the `print` **function**.

```
var Person[] a = { new Person(STRINGBGFG("P # 1")), new Person(STRINGBGFG("P # 2")),
new Person(STRINGBGFG("P # 3"))};
print(a);
```

Question 10.3: Write your own **class** called Mine similar to the Person **class** with a one int parameter **constructor**, a private int property `p` and a `toString` **method** that converts `p` to a string. Then write a **function** in the PersonTest **class** with same name as the previous `print` **function**, except that this one takes a Mine`[]`, also known as a Mine array. You might recall from Tutorial 7 that this practice of having two functions with the same name is called **function** *name overloading*. Change the **main** **function** to what follows so that it initializes an array of Mine point variables and then calls the second `print` **function**.

```
var Mine[] b = { new Mine(1), new Mine(2), new Mine(3) };
print(b);
```

Here is an example of a second initialization syntax. For this particular example it is better to use the simpler, earlier initialization syntax, but when the size of the array to be created is to be determined at run-time, then the second syntax should used. The next question will show you an example of this.

```
beginMain
   var Person[] a = new Person[3];
   // at this point the array is all nulls
   for (var int i=0; i<3; i=i+1)

      a[i] = new Person(STRINGBGFG("P # ") + (i+1));

   print(a);
endMain
```

Question 10.4: Write a **function** `create` takes one `int` argument, the size of the array to create and returns a `Person` array of that size. Make it so the i^{th} element of the array is initialized to `STRINGBGFG("P # ") + i`. Call this **function** from the **main function** like so:

```
beginMain
   var Person[] a = create(3);
   print(a);
endMain
```

Question 10.5: Write a **function** `create2` takes one int argument, the size of the array to create and returns a `Mine` array of that size. Make it so the i^{th} element of the array's **`toString` method** prints out `STRINGBGFG("Mine # ") + i`. Why is it not possible to overload that **`create` function**? Try it and see what the compiler says. Call `create2` from the **main function** like so:

```
beginMain
   var Mine[] a = create2(3);
   print(a);
endMain
```

Question 10.6: Write a **function** `doubler` that takes a `Person` array `x` and returns a new `Person` array called `result` twice as big as `x`. Copy `x` into the `result` before you return it. The extra elements in `result` should all be `null`.
Question 10.7: Change the **`doubler` function** so that every `null` in the array `result` is set to a new `Person` make it so that every new `Person` object has a different `name` property.

Two dimensional arrays

Question 10.8: Here is an example of a convenient two dimensional array initialization syntax. Study, compile and run the following code. The code `Person[][]` should be read out loud as *person array array* indicating the variable `a` is a *person array array*, also known as a *two-dimensional array of persons*.

```
beginMain
   var Person[][] a = { { new Person(STRINGBGFG("P # 1")), new Person(STRINGBGFG("P #
2")), new Person(STRINGBGFG("P # 3")) },
                        { new Person(STRINGBGFG("P # 4")), new Person(STRINGBGFG("P # 5"))
},
                        { new Person(STRINGBGFG("P # 6")) } };

   for (var int y=0; y<a.length; y=y+1)

      for (var int x=0; x<a[y].length; x=x+1)

         System.out.print(STRINGBGFG(" ") + a[y][x]);

      System.out.println();

endMain
```

Question 10.9: By copying the pattern of the code above, do some more overloading of the `print` **function** by writing two new `print` functions, one taking a two dimensional array of `Person`, the

other taken a two dimensional array of Mine. The call both of these functions from the **main function**.
Since `a[0]` is a Person array,you would naively expect it to be able to be re-initialized like so:

```
a[0] = { new Person(STRINGBGFG("P # 4")),
         new Person(STRINGBGFG("P # 5")),
         new Person(STRINGBGFG("P # 6")) };
```

so that after this code `a0` holds the four element long array `Person # 4`,`Person # 5` and `Person # 6`,but it does't work owing to a design oversight by the creators of Java. Luckily as mentioned above there is a way around this oversight and that is to use a temporary variable like so:

```
var Person[] temp = { new Person(STRINGBGFG("P # 4")),
                      new Person(STRINGBGFG("P # 5")),
                      new Person(STRINGBGFG("P # 6")) };
a[0] = temp; // Array "a[0]" now holds P # 4,P # 5,P # 6
```

Like with one dimensional arrays,there is a second initialisation syntax for two dimensional arrays and here it is. Unlike the above code the sub-arrays `a[0]`,`a[1]` and `a[2]` are all of equal size,namely three.

```
var Person[][] a = new Person[3][3];
a[0][0] = new Person(STRINGBGFG("P # 1"));
a[0][1] = new Person(STRINGBGFG("P # 2"));
a[0][2] = new Person(STRINGBGFG("P # 3"));
a[1][0] = new Person(STRINGBGFG("P # 4"));
a[1][1] = new Person(STRINGBGFG("P # 5"));
a[1][2] = new Person(STRINGBGFG("P # 6"));
a[2][0] = new Person(STRINGBGFG("P # 7"));
a[2][1] = new Person(STRINGBGFG("P # 8"));
a[2][2] = new Person(STRINGBGFG("P # 9"));
```

Question 10.10: Write a **function** `create3` and `create4` that takes an int argument `size` and returns a two dimensional array of Person or Mine, respectively. Make is so that each Person or Mine object has its own number, using a separate counter variable int `count`.

Three dimensional arrays

Question 10.11: Using the knowledge you have gained so far about arrays, create, initialize and print a three dimensional array of Person. Make it so that each Person object is given its own number using a separate counter variable int `count`.

2.6.11 Tutorial 11

The following code presents example involving three classes Flea, Dog and DogOwner to represent the idea that a *dog* has a *flea* and a *dog-owner* has a *dog*. The **class** DogTest is the driver **class**. The key concept of this tutorial is that classes can have references of objects of another **class** in order to set up a relationship between the two classes.
Question 11.1 Study the following code and find the two bugs in it. Fix the bugs and then compile and run it to verify that it prints out STRINGBGFG(`"p=I am a flea called Pop"`).

```
// BEGIN FILE: jtw-tutorials/DogTest.jtw
class Flea
begin
   property String name;

   constructor Flea(String aName)
   begin
      aName = name;
   end

   public method String toString()
   begin
      return "(I am a flea called " + name + ")";
   end
end

class Dog
begin
   property String name;
   property int    age;      // Age in years
   property Flea   dogsFlea;

   constructor Turtle(String aName, int anAge, Flea aFlea)
   begin
      name     = aName;
      age      = anAge;
      dogsFlea = aFlea;
   end
end

class DogTest
begin
   beginMain
      var Flea p = new Flea("Pop");
      var Flea s = new Flea("Squeak");
      var Flea z = new Flea("Zip");
      System.out.println("p=" + p);
   endMain
end
// END FILE:    jtw-tutorials/DogTest.jtw
```

Question 11.2: In the **main function** of the DogTest **class**, write code to call the `toString` **method** for the fleas referenced by `s` and `z`.

Question 11.3: In the **main method** of the DogTest **class**, write code to construct three dogs called `"Fido"`, `"Jimbo"` and `"Rex"`. For the purposes of the rest of these questions, let the name of the references for Fido, Jimbo and Rex be `f` `j` and `r`. Note that the third parameter to the Dog **class** is of type Flea. Therefore you will need to supply a Flea reference for each dog. Make it so that Fido has a flea called Pop, Jimbo has a flea called Squeak, and Rex has a flea called Zip.

HINT: If the flea called Pop is referenced by the variable name `p`, then this reference should appear as the third argument in one of the calls to the Dog **constructor**.

Question 11.4: Write a `toString` **method** in the Dog **class** that works like the `toString` **method** in the Flea **class**. Then call this **method** from the **main function** to print out the full statistics of the three dogs that you have just created in Question 11.3.

Question 11.5: By copying the pattern of the Flea and Dog classes, write a **class** DogOwner that has three non-**private** properties: `name`, `salary` and `ownersDog`. Also write a three-parameter **constructor** for the DogOwner **class** that sets these properties.

Question 11.6: Add some code into the **main function** to construct three dog owners called `Angus`, `Brian` and `Charles`. Make it so that Angus has a dog called Rex, Brian has a dog called Jimbo, and Charles has a dog called Fido. For the purposes of the rest of these questions, let the name of the references for Angus, Brian and Charles be (respectively) `a`, `b` and `c`. Use the Dog

references that you created in Question 11.3 to achieve this. Make it so that Angus, Brian and Charles have initial salaries of 10,000, 20,000 and 30,000.

Question 11.7: Without changing the call to the DogOwner **constructor**, change the value of the `salary` property of object referenced by `a` to 1,000,000. Note that since the `salary` property of the DogOwner **class** is non-**private** you should be able to set the value of the `salary` property from the **main function** of DogTest.

Question 11.8: Write a `toString` **method** for the **class** DogOwner and add some code to the **main function** to call it for Angus, Brian and Charles.

Question 11.9: What is the value of: `a.ownersDog.dogsFlea.toString()`? Add some code to the **main function** to find out if it does what you think it should do.

2.6.12 Tutorial 12

Question 12.1: Write **constructor**s for the classes SportsShoe and Runner below, by looking at the **main function** to see how many arguments each **constructor** has.

```
// BEGIN FILE: jtw-tutorials/RunnerTest.jtw
class SportsShoe
begin

   property String model;        // what kind of shoe it is
   property double speedBoost;   // the boosting factor of the shoe

   // constructor goes here:

   // Useful method for debugging
   method String toString()
   begin
      return "(I am a shoe called " + model + " and my boosting factor is " + speedBoost + ")";
   end

end

class Runner
begin
   private property String     name;     // Runner's name.
   private property int        speed;    // speed of runner in km/hr.
   private property SportsShoe shoes;    // which shoe they are wearing

   // constructor goes here:

   // Useful method for debugging
   method String toString()
   begin
      return "(I am a runner and my name is " + name + " and my shoes are " + shoes + ")";
   end

   /*
   ** This private method computeSpeed works out the runners speed,
   ** based on their basic speed and the speed boost due to the
   ** SportsShoe that they are currently wearing.
   */

   // method goes here:


   /**
   ** Prints the result of racing two runners against each other.
   */
   function void race(Runner r1, Runner r2)
   begin
      if (r1.computeSpeed() > r2.computeSpeed()) then
```

```
        begin
            System.out.println("Runner " + r1.name + " beats " + r2.name);
        end
        else
        begin
            System.out.println("Runner " + r2.name + " beats " + r1.name);
        end
    end


    /**
     ** Swaps the shoes of two runners.
     */
    function void swapShoes(Runner r1, Runner r2)
    begin
      var SportsShoe tempShoe = r1.shoes;
      r1.shoes = r2.shoes;
      r2.shoes = tempShoe;
  end
end

class RunnerTest
begin
    beginMain
        var SportsShoe nike    = new SportsShoe("Nike NX-71",   2.0);
        var SportsShoe reebock = new SportsShoe("Reebock R20",  2.3);
        var SportsShoe puma    = new SportsShoe("Puma P200-MMX",4.8);

        var Runner sg = new Runner("Speedy Gonzalez", 55, nike);
        var Runner sw = new Runner("Slick Willy",     49, reebock);
        var Runner fa = new Runner("Fat Albert",      15, puma);

        Runner.race(sg,sw);
        // Runner.race(sg,sw,fa);
        // sg.raceAgainst(sw);
    endMain
end

// END FILE:    jtw-tutorials/RunnerTest.jtw
```

Question 12.2: In the Runner **class**, write the private **method** `computeSpeed` that has no arguments and returns a double-precision floating point value that equals the runner's running speed. Note that the speed of a runner is determined by multiplying their `speed` property with the `speedBoost` property of the shoes that they are wearing. For example, Speedy Gonzalez's running speed = 55 * 2.0 = 110.0.

Question 12.3: Fix the `race` **method** so that it checks for a draw.

Question 12.4: By copying the `race` **method**, write a three-parameter `race` **method** for racing three runners against each other. Two methods in the same **class** with the same name is called *overloading* in Java. Add a call to this **method** from the **main function**.

Question 12.5: What is the difference between a **method** and a **function**? Write a one parameter **method** `raceAgainst` that behaves exactly like two-parameter **function** `race`. There are two ways of doing this, one is to optionally use the **this** keyword rather than one of the parameters `r1` or `r2`. The second way is for `race` to simply call `race` using **this** as one of the arguments to the **function**.

Question 12.6: Is it true that any **method** can be re-worked into a **function** and vice versa?

Question 12.7: The `swapShoes` **method** in the Runner **class** swaps the shoes of two runners. Add some code to the **main function** to swap the shoes of two runners and verify that the shoes do indeed get swapped.

Question 12.8: Write a **method** called `swapNames` that swaps the names of two runners. You

can put this **function** into any **class** but it makes the most sense to put it into the Runner **class** since it has two Runner parameters.
Question 12.9: Write a **method** `swapSpeeds` that swaps the `speed` properties of two runners.

2.6.13 Tutorial 13

Question 13.1: Study, compile and run the following code:

```
// BEGIN FILE: jtw-tutorials/CarTest.jtw
class Car
begin

   property String      model;
   property int         value; // Car's value in dollars
   property int         serialNumber;
   private classVar int serialCounter = 1000;

   constructor Car(String aModel, int aValue)
   begin
      model         = aModel;
      value         = aValue;
      serialNumber  = serialCounter;
      serialCounter = serialCounter + 1;
   end

   public method String toString()
   begin
      return "(I am a car, model=" + model + ", value=" + value +
             ", serial number=" + serialNumber + ")";
   end
end

class Owner
begin

   property String name;
   property int    money; // Owner's money in dollars
   property Car    ownersCar;

   constructor Owner(String aName, int aMoney, Car aCar)
   begin
      name      = aName;
      money     = aMoney;
      ownersCar = aCar;
   end

   public method String toString()
   begin
      return "(I am a car owner, name=" + name + ", money=" + money +
             ", car=(" ) + ownersCar + ")";
   end
end

class CarTest
begin
   beginMain
      var Car   ford   = new Car "Ford Escort",1000);
      var Car   nissan = new Car "Nissan Nivara",2000);
      var Owner joe    = new Owner "Joe Bloggs",500,ford);
      var Owner mary   = new Owner "Mary Smith",600,null); // Mary has no car to start with.
      joe.describe();
```

```
   endMain
end
// END FILE:    jtw-tutorials/CarTest.jtw
```

Question 13.2: What is the purpose of the **class** variable `serialCounter`?
Question 13.3: Write a **method** `sellCar` that increases the owner's money by half the value of their car and the owner's car reference gets set to `null`, for no car. If the owner owns no car (`null`) simply do nothing.
Question 13.4: Write a **method** in the Owner **class** called `purchase` so that:

```
Car newCar = new Car(STRINGBGFG("Mini Cooper"),1000);
joe.purchase(newCar);
```

results in Joe's money going down by `newCar.value` and Joe's car being set to `newCar`. Call the `sellCar` method before Joe purchases his new car
Question 13.5: Write a **function** in the Owner **class** called `netWorth` so that:

```
System.out.println(STRINGBGFG("Joe's net worth = ") + joe.netWorth());
```

prints out Joes' money plus the value of his car, if he has a car. You will need to use an **if** (...) **then** ... statement to test whether or not a reference is pointing to a valid object or `null` for no object like so:

```
if (ownersCar == null)
then
   // do not access ownersCar.value as ownersCar points to no object

else
   // do access ownersCar.value
```

Question 13.6: Write a **method** in the Owner **class** called `smashCar` so that:

```
mary.smashCar();
```

halves the value of Mary's car.
Question 13.7: Write a **method** in the Owner **class** called `stealCarFrom` so that:

```
mary.stealCarFrom(joe);
```

results in Mary selling his current car (if he has one) for its market value and Mary acquiring ownership of Joe's car. Also make Joe invoke his `sellCar` method to relinquish ownership of his car if he has one.
Question 13.8: Write a **function** in the Owner **class** called `swapMoney` so that:

```
Owner.swapMoney(joe,mary);
```

swaps the money of Joe and Mary.
Question 13.9: Write a **function** in the Owner **class** called `swapCars` so that:

```
Owner.swapCars(joe,mary);
```

swaps the cars of Joe and Mary.
Question 13.10: Write a **function** in the Car **class** called `swapSerialNumbers` so that:

```
Car.swapSerialNumbers(ford,nissan);
```

swaps the serial numbers of `ford` and `nissan`.
Question 13.11: Write a **function** in the Owner **class** called `sellCarTo` so that

```
joe.sellCarTo(mary);
```

results in Joe's money going up by the value of his car and Mary's money going down by the value of his car, and the ownership of Mary's car gets transferred to Joe.

2.6.14 Tutorial 14

Dr Seuss' story [*Yertle the Turtle*] describes how a turtle called Yertle sits at the top of a pile of other turtles. In this example, the pile of turtles is represented by a linked list of Turtle objects, with the `down` property serving to connect one Turtle object to another. If a Turtle object has a non-`null` `down` property, then this represents a turtle that is sitting below the current one. The last turtle in the linked list is the turtle that is at the bottom of the pile, which has a `null` value for its `down` property.
Question 14.1: Study, compile and run the following code:

```
// BEGIN FILE: jtw-tutorials/TurtleTest.jtw
package files;

class Turtle
begin

   private property String  name;
   private property int     age;    // Turtle's age in years
   private property double  weight; // Turtle's weight in kg

   // NOTE: this property allows for linked lists
   property Turtle down;

   constructor Turtle(String aName, int anAge, double aWeight)
   begin
      name   = aName;
      age    = anAge;
      weight = aWeight;
   end

   /** Getter method for name property */
   method String getName()
   begin
      return name;
   end

   /** Getter method for weight property */
   method double getWeight()
   begin
      return weight;
   end
```

```
   /** Useful method for debugging */
   public method String toString()
   begin
      return name;
   end

   /** Inserts the turtle t below the current one */
   method void insert(Turtle t)
   begin
      var Turtle temp = this.down;
      this.down = t;
      t.down = temp;
   end
end

public class TurtleTest
begin
   beginMain

      var Turtle yurtle = new Turtle("Yurtle", 103, 20);
      var Turtle zippy  = new Turtle("Zippy",  102, 30);
      var Turtle bungle = new Turtle("Bungle", 101, 40);

      // *** see later
      yurtle.down = zippy;
      zippy.down  = bungle;
      bungle.down = null; // NOTE: not needed as bungle.down is null by default

      var int totalWeight = 0;
      for (var Turtle current = yurtle; current != null; current=current.down)
      begin
         totalWeight = totalWeight + current.getWeight();
      end
      System.out.println("The total weight is " + totalWeight);
   endMain
end
// END FILE:    jtw-tutorials/TurtleTest.jtw
```

The code in the **main function** after the *** sets up the following relationships between the three Turtle objects (Bungle, Zippy and Yertle). Figure 2.2 shows the relationship between the different turtles. When you traverse the list of turtles you must always start at the top turtle (known as the *head of the linked list*). If you give a different value for the top turtle, your code will think that the given turtle is the one at the top of the pile and you will get the wrong result.

Question 14.2: Move the code for calculating the total weight of the turtles from the **main function** to a **function** called **function** void **printTotalWeight**(Turtle top) in the Turtle **class** that prints out the total weight of the turtles. Then call that **function** from the **main function** to get the same result as before. Note that that if `printTotalWeight` was a **method** then calling that **method** using `null` (representing an empty list) like so: `null.printTotalWeight()` would be an error, whereas `Turtle.printTotalWeight(null)` wouldn't be and therefore is better. This is one example of how methods and functions differ.

Question 14.3: Revision question for getters. By copying the pattern established by the `getName` **method**, add two getter methods to the Turtle **class**: `getAge` which returns the current turtle's age and `getWeight` which returns the current turtle's weight. Then call these methods on the Yertle object in the **main function**. Note that the `toString` **method** would be more appropriate as it handles nulls better but you known that the `yurtle` reference is not `null` so you know it is safe to call the `getAge` and `getWeight` methods on the `yurtle` reference.

Question 14.4: Write a **function** Turtle **findBottomTurtle**(Turtle top) that returns the Turtle object that is at the top of the pile, and returns `null` if there isn't one.

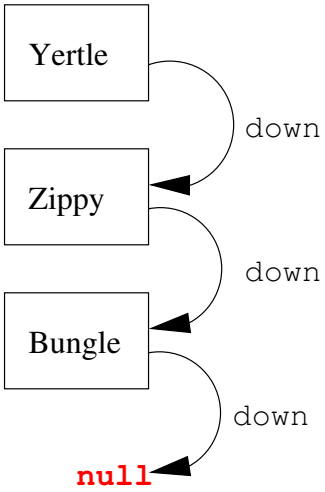


Figure 2.2: A linked list of `Turtle` objects

Question 14.5: Then call this **function** from the **main** **function** using **System.out.println**() and the top turtle `yertle`.
Question 14.6: Write a **function** Turtle **findOldestTurtle** (Turtle `top`) that returns the oldest turtle or `null` if there isn't one.
Question 14.7: Then call this **function** from the **main** **function** using **System.out.println**() and the top turtle `yurtle`.
Question 14.8: Write a **function** Turtle **findHeaviestTurtle** (Turtle `top`) returns the heaviest turtle, or `null` if there isn't one.
Question 14.9: Then call this **function** from the **main** **function** using **System.out.println**() and the top turtle `yurtle`.
Question 14.10: Write a **function** void **sayPile** (Turtle `top`) that prints the names of the turtles in the pile starting from the top turtle and finishing at the bottom turtle. Then call this **function** from the **main** **function**.
Question 14.11: Under what circumstances would it be okay to change the visibility of the `down` property to private, like the `name`, `age` and `weight` properties?
Question 14.11: Add an extra parameter to the **constructor** which is a reference the to the turtle below of the current one. Then remove all occurrences of the `down` property from the **main** **function**. The advantage of this is that it enables you to change the visibility of the `down` property to **private**.

2.6.15 Tutorial 15

Basic Inheritance

When you see the following code: **class** X **extends** Y, it means that **class** X *inherits* from the **class** Y. Class X is called the *subclass* and the **class** Y is called the *super-class* or sometimes the *parent* **class**. When the **class** X extends from Y, it pulls in all of the non-**private** **method**s and **property**s from the super-**class** Y. Inherited **method**s can override the behaviour of that same **method** in the super-**class** to give behaviour that is specific to the sub-**class**. The concept of **method**s overriding other **method**s is called *dynamic* **method** *binding* or more commonly the more impressive-sounding name: *polymorphism*. The main thing that this tutorial shows is the idea that inheritance is a non-symmetrical relationship. For example: in the code that follows, the Bird **class** inherits from the Animal **class**, which corresponds to the idea that *every bird is an animal*. The reverse, *every animal is a bird* is plainly not true! Inheritance forces you to recognize this.

Question 15.1: Study, compile and run the following code. The following code shows how inheritance works. In the following code, the Bird **class** inherits from the Animal **class**. The Bird **class** pulls in the Animal **class**'s `age` property and the `canFly` and `talk` **method**s. Importantly the `canFly` property overrides the behaviour of the `canFly` **method** of the parent Animal **class**, which reflects that fact that generally speaking, birds can fly. In the code that follows, note that *int* properties are initialized to zero by default and the *super* **method** (also known as the **constructor** of the super-class) is called by default if there is a zero parameter **constructor** in the super-class, which there is by default, even if you don't write one!

```
class Animal


   property int age; // Animal's age in years
   property int health; // Animal's health in hit points

   constructor Animal()

      age = 0; // NOTE: not needed as set by default
      health = 100;


   method boolean canFly ()

      return false;


   method void talk ()

      System.out.println(STRINGBGFG("Hello"));



class Bird extends Animal


   property double flySpeed; // Bird's speed in km/h

   constructor Bird()

      super();      // NOTE: not needed as called by default
      flySpeed = 0; // NOTE: not needed as set by default


   method boolean canFly ()

      return true;


   method void peck ()

      System.out.println(STRINGBGFG("peck"));
```

```
class InheriTest

   beginMain
      var Bird eagle = new Bird();
      eagle.talk();
      eagle.peck();
   endMain
```

Question 15.2: Override the `talk` **method** of the Animal **class** in the Bird **class** to print out `STRINGBGFG("Tweet Tweet!")` rather than `STRINGBGFG("hello")` to give more accurate talking of bird objects.
Question 15.3: By copying the pattern established in the Bird **class**, change the eagle from an instance of the Bird **class** to its own **class** in its own right and then create an instance of that **class** in the **main** **function** of InheriTest. Your Eagle **class** should have one **property**: `int numberOfKills` and one **method**: `void attack()` that internally increments the value of `numberOfKills`. In the **main** **function** you should call every **method** of the Eagle **class** and its super-classes.
Question 15.4: What is the advantage of using a new separate **class** to represent a new object rather than using an instance of an existing **class**?
Question 15.5: Create a new **class** Kiwi that inherits from the Bird **class**. Your Kiwi **class** should override the `canFly` **method** to return false, which reflects the fact that generally speaking birds can fly, but the kiwi bird in particular does not fly. Your Kiwi **class** have a property `numberOfWorms`. Once you have written the Kiwi **class** you should create an instance of the Kiwi **class** in the **main** **function**.
Question 15.6: Why does the following line of code in the **main** **function** print out 100 but there is no setting of that variable to that value in the Kiwi **class**?

```
System.out.println(k.health);
```

Question 15.7: In the classes Animal, Bird, Eagle and Kiwi, remove all of the `canFly` **method**s and replace it with a single `canFly` property of the Animal **class**. In the **constructor**s you will need to set the value of the `canFly` **property** to a value that is appropriate for that **class**. For example in the Bird **class**'s **constructor** you should set the `canFly` property to true, while in the Kiwi **class**'s **constructor** you should set the `canFly` **property** to false.
Question 15.8: What is the advantage of having a `canFly` **property** over a bunch of `canFly` **method**s?
There is an equally valid alternative to having a **public property** in the Animal **class** and that is to have in the Animal **class** a **private property** `canFly` and a pair of **method**s for getting and setting the value of the `canFly` property like so. These **method**s in J.T.W. and Java are called *getter* **method**s and *setter* **method**s since, as their names suggest, getters are used for getting the value of something and setters are used for setting the value of something. Note that the `canFly` **method** of the code above corresponds to `getCanFly` **method** in the code below.

```
private property boolean canFly;

method boolean getCanFly ()

   return canFly;


method void setCanFly (boolean aCanFly)
```

```
    canFly = aCanFly;
```

You might think that it is simpler to have one thing (a single non-**private property**) rather than three things (a **private** property and a non-**private** getter **method** and a non-**private** setter **method**) and you would be right. However from the point of view of the client code that uses the Animal **class**, the two approaches are identical. Later on when you learn more you will understand under what circumstances the second getter and setter approach is better.

Question 15.9: Change the **main** **function** to what follows:

```
var Bird b = new Bird(10);
var Animal a = b;
a.talk();
a.peck();
```

When you compile this code it gives a compilation error. What line gives the error and what is the reason for the error?

Question 15.10: Change the **main** **function** to what follows:

```
var Animal a = new Animal();
var Bird b = a;
b.talk();
b.peck();
```

When you compile this code it gives a compilation error. What line gives the error and what is the reason for the error?

Run-time type inquiry

In J.T.W. and Java there is a keyword called *instanceof* that does a run-time check on the type of an object. The following **function**:

```
function void say (Animal a)

  System.out.println(a instanceof Bird);
```

uses the **instanceof** keyword to determine the run-time type of the reference `a` and prints out whether or not the reference is referring to a Bird object. Some examples should clarify the situation:

- `say(new Bird())` prints `true`, Since the parameter `a` is pointing to a bird object at run-time,
- `say(new Animal())` prints `false` since not every animal is a bird,
- `say(new Eagle())` prints `true`, since every eagle is a bird, and
- `say(new Kiwi())` prints `true`, since every kiwi is a bird.
- `var Animal a = new Animal(); say(a);` prints `false` since at run-time `a` is not pointing to a bird object

- **var** Animal a = **new** Bird(); say(a); prints true since at run-time a is pointing to a bird object.

In Tutorial 17 you will learn why in most cases it is better to use polymorphism instead of the **instanceof** keyword for run-time type enquiry.

The super-class of all objects

Every **class** in Java inherits either directly or indirectly from a **class** called *Object*. That is to say if x is a reference variable, then the run-time expression x instanceof Object is always true except for the pathological case where x is null (i.e. is currently pointing to *no object*). The Object **class** contains a **method** called toString that returns a string containing the run-time **class** name of the object concatenated with the something like the memory address of the object in base 16 (also known as *hexadecimal*) format. Since every **class** inherits from Object, every object can have toString invoked upon it. Even better, every **class** X can override toString to provide debugging information that is tailored to X. Therefore the toString **method** is convenient for debugging. Since the toString **method** is a **public method** of the Object **class** it must be overridden as a **public method**, since your overridden **function** cannot have weaker access privileges.

2.6.16 Tutorial 16

This tutorial shows you a practical example of inheritance. The file StarWars.jtw is comprised of three classes: XWing, TieFighter and StarWars. The first two represent spacecraft from the two sides of the Star Wars films. The **class** StarWars is the driver **class** and contains code for executing a battle between the X-Wings and the Tie Fighters.
Question 16.1: Study, compile and run the following code:

```
class XWing


   private property int shields;
   private property int weapon;
   private property boolean dead;

   constructor XWing()

      shields = 1000;
      weapon = 10;


   method int getWeapon()

      return weapon;

   method boolean isDead()

      return dead;

   method void hit(int damage)

      shields = shields - damage;
      if (shields<0)
      then
```

```
            System.out.println(STRINGBGFG("BOOM!!!"));
            dead = true;



class TieFighter


   private property int shields;
   private property int weapon;
   private property boolean dead;

   constructor TieFighter()

      shields = 500;
      weapon = 20;


   method int getWeapon()

      return weapon;

   method boolean isDead()

      return dead;

   method void hit(int damage)

      shields = shields - damage;
      if (shields<0)
      then
         System.out.println(STRINGBGFG("BOOM!!!"));
         dead = true;



class StarWars


   private function void duel(XWing x, TieFighter t)


      for (;;)

         x.hit(t.getWeapon());
         if (x.isDead())
         then
            System.out.println(STRINGBGFG("X-Wing is dead"));
            break;

         t.hit(x.getWeapon());
         if (t.isDead())
```

```
            then
               System.out.println(STRINGBGFG("Tie Fighter is dead"));
               break;




      private function void battle (XWing[] good, TieFighter[] evil)


         var int g         = 0;
         var int e         = 0;
         var int goodDeaths = 0;
         var int evilDeaths = 0;

         while (g<good.length and e<evil.length)

            System.out.println(STRINGBGFG("battling X-Wing #") + g + STRINGBGFG(" versus Tie
Fighter #") + e);
            duel(good[g],evil[e]);
            if (good[g].isDead())
            then
               g = g + 1;
               goodDeaths = goodDeaths + 1;

            if (evil[e].isDead())
            then
               e = e + 1;
               evilDeaths = evilDeaths + 1;



         var int finalGood = good.length - goodDeaths;
         var int finalEvil = evil.length - evilDeaths;

         System.out.println();
         System.out.println(STRINGBGFG("Battle Report:  X-Wings Tie Fighters"));
         System.out.println(STRINGBGFG("-------------------------------------------"));
         System.out.println();
         System.out.println(STRINGBGFG("Initial ships:") + good.length + STRINGBGFG(" ") + evil.length);
         System.out.println();
         System.out.println(STRINGBGFG("Killed ships:") + goodDeaths + STRINGBGFG(" ") + evilDeaths);
         System.out.println();
         System.out.println(STRINGBGFG("Final ships:")   + finalGood + STRINGBGFG(" ") + finalEvil);
         System.out.println();
         if (finalGood>finalEvil)
         then
            System.out.println(STRINGBGFG("The rebel alliance is victorious!"));

         else
            System.out.println(STRINGBGFG("The dark side has conquered!"));

         System.out.println();
```

```
beginMain

   // defines the goodies array
   var XWing[] goodies = new XWing[3];

   // initializes the elements of the goodies array
   for (var int i=0; i<goodies.length; i = i + 1)

      goodies[i] = new XWing();


   // defines the baddies array
   var TieFighter[] baddies = new TieFighter[3];

   // initializes the elements of the baddies array
   for (var int i=0; i<baddies.length; i=i+1)

      baddies[i] = new TieFighter();


   battle(goodies,baddies);

endMain
```

Question 16.2: Compile and run this file to see the battle between the X-Wings and the Tie Fighters unfold.
Question 16.3: If you look at the Java code for the XWing and TieFighter classes you will notice that they are almost identical: They have the same methods and properties, the only difference is that the XWing objects are initialized with a different value for their shields and weapon properties to the TieFighter objects.
The next few questions will guide you through the process of using inheritance to eliminate this unnecessary duplication of code. A new **class** called SpaceShip will be created and all of the code that is common to XWing and TieFighter will be moved into this **class**. The XWing and TieFighter classes will then be modified so that they both inherit from SpaceShip.
Question 16.4: The first step in this process is to create the outer shell of the SpaceShip **class**, which you should now type in:

```
class SpaceShip
```

Question 16.5: Move the properties `shields`, `weapon` and `dead` out of the XWing and TieFighter classes and into the SpaceShip **class**. You must change the privacy status of the properties from *private* to *protected*. The protected modifier was invented as an intermediate level of privacy between public and private. Like **private**, it allows visibility to the same **class** in which the **method** or **property** was defined, but unlike **private** it also allows visibility to sub-classes of the **class** in which the **method** or **property** was defined.
Question 16.6: Move the three methods `getWeapon`, `isDead` and `hit` out of the XWing and TieFighter classses and into the SpaceShip **class**. At this point, the XWing and TieFighter classes should contain nothing but a **constructor**.

Question 16.7: Finally, add the *extends* keyword to the first line of the XWing and TieFighter classes:

```
class XWing extends SpaceShip
```

and

```
class TieFighter extends SpaceShip
```

Question 16.8: Compile and run your program again, making sure that it produces the same results now that it is using inheritance.
Question 16.9: The SpaceShip **class** is a superclass of both XWing and TieFighter containing everything that X-Wings and Tie Fighters contain in common. Because the role of the SpaceShip **class** is simply to hold these commonalities, we might choose to label the **class** with the **abstract** keyword:

```
abstract class SpaceShip
```

This prevents us from creating instances of the SpaceShip **class**. Without the **abstract** modifier, we could happily create a **new** SpaceShip(), which would be an object that is not an X-Wing, nor a Tie Fighter, but just a vague "space ship". If we consider this to be a logical mistake then we can use **abstract** to prevent such calls to the SpaceShip **constructor**. Change the **class** SpaceShip to be **abstract** and observe how the compiler will not accept any lines of the form:

```
var SpaceShip s = new SpaceShip(); // compiler error
```

Remove the **abstract** keyword and notice how the compiler will then allow this line to compile.

2.6.17 Tutorial 17

Question 17.1: Study the following code:

```
class AnimalTest

   private function void chatter (Animal[] a)

      for (var int i=0; i<a.length; i=i+1)

         a[i].talk();


   beginMain
      var Animal[] farm = { new Dog(),new Cow(),new Fish() };
      var Animal[] ark = { new Dog(),new Dog(),new Cow(),new Cow(),new Fish(), new Fish() };
      var Cow[]   herd = { new Cow(),new Cow(),new Cow() };
      chatter(farm);
      chatter(ark);
      chatter(herd);
   endMain


class Animal
```

```
    method boolean breathesUnderwater()
        return false;

    method boolean isPredator()
        return false;

    method void talk()


class Dog extends Animal
    method boolean isPredator()
        return true;

    method void talk()
        System.out.println(STRINGBGFG("Woof woof!"));
```

Question 17.2: Write the following classes that sub-**class** the Animal **class** above: Cow, Cat, Fish, and Whale.
Question 17.3: Write the Shark **class** which extends Fish **class**. Override all necessary **method**s. For the sake of this example and the code that follows, suppose that shark's `talk` **method** prints out `STRINGBGFG("Chomp Chomp!")`.
Question 17.4: Run the AnimalTest **class** to make sure that all the **method**s work correctly.
Question 17.5: Rewrite the `chatter` **method** so that it never calls the `talk` **method**s and instead uses a series of *if* statements and the *instanceof* operator to test the run-time type of each object in the `a` array. Here is some code to get you started:

```
private function void chatter(Animal[] a)
    for (var int i=0; i<a.length; i=i+1)
        if (a[i] instanceof Cow) then
            System.out.println(STRINGBGFG("Moo!"));
        else if (a[i] instanceof Cat) then
            System.out.println(STRINGBGFG("Meow!"));
        /* other code goes here */
```

Note that the sub-**class**es must appear before super-classes in the above code, otherwise the wrong message will be printed out for sub-**class**es.
Question 17.6: Why is the code from the last question not as good as calling each animal's `talk` **method**?

2.7 Proofs of concept for the J.T.W language

2.7.1 Proof of concept #1: A small collection of d-defmacros for your use in client code

Study the following Elisp code which creates a pair of macros `getter` and `setter`, a macro for implementing the *singleton design pattern* called `singleton_design_pattern` and a macro `foreach` for implementing the iterator design pattern.

```
;; BEGIN FILE: ~/dlisp/d-defmacro.el
;;; d-defmacro.el

;; Copyright (C) 2017 Davin Pearson

;; Emacs Lisp Archive Entry
;; Filename: d-defmacro.el
;; Author/Maintainer: Davin Max Pearson <http://davin.50webs.com>
;; Keywords: defmacros for defining macros in J.T.W.
;; Version: 1.0

;;; Commentary:

;; This file is part of GNU Java Training Wheels.
;;
;;; m4_limitation_of_warranty

;;; m4_install_instructions (d-defmacro)

;;; Known Bugs:

;; None so far!

;;; Code:

;;(load-file "~/lisp++-projects/c++2lisp++-stage-1-purge-comments.el")
;;(load-file (concat (car load-path) "lisp++-mode.el"))
;;(load-file "~/lisp++-projects/lisp++2c++-cclass.el")

(safe-require 'd-flm)

(setq d-macro-list nil)

(defmacro d-defmacro (name &rest macro-form)
  `(progn
     (setq d-macro-list (cons (quote , name) d-macro-list))
     (defmacro , name (&rest rest)
       ,@ macro-form
          )
     ))

;;(setq type "int")
;;(setq vari "v")
;;(setter int i)
```

```
(d-defmacro
 getter
 (setq type (nth 0 rest))
 (setq vari (nth 1 rest))
 (d-assert (cdr rest))
 (d-assert (not (cdddr rest)))
 (if (not (stringp type))
     (setq type (prin1-to-string type)))
 (if (not (stringp vari))
     (setq vari (prin1-to-string vari)))
 (setq prop nil)
 (setq var (d-read-str (concat "getter-setter-prop--" type "--" vari)))
 (when (not (and (boundp var) var))
   (set var t)
   (setq prop (concat "private " type " private_" vari ";")))
 (concat "public " type " get" (d-string-capitalise vari) "() "
         "{ return private_" vari "; }" prop "\n"))

(d-defmacro
 setter
 (setq type (nth 0 rest))
 (setq vari (nth 1 rest))
 (d-assert (cdr rest))
 (d-assert (not (cdddr rest)))
 (if (not (stringp type))
     (setq type (prin1-to-string type)))
 (if (not (stringp vari))
     (setq vari (prin1-to-string vari)))
 (setq prop nil)
 (setq var (d-read-str (concat "getter-setter-prop--" type "--" vari)))
 (when (not (and (boundp var) var))
   (set var t)
   (setq prop (concat "private " type " private_" vari ";")))
 (concat "public void set" (d-string-capitalise vari) "(" type " " vari ")"
         " { this.private" vari " = " vari "; }" prop "\n"))

;; (d-compress-args '("100" "200" "300" ")"))
(defun d-compress-args (rest)
  (let ((ptr    rest)
        (result "(")
        (count  0)) ;; (setq count 0)
    (while ptr
      (when (not (string= (car ptr) ")"))
        (setq result (concat result (if (/= count 0) "," ) (car ptr)))
        (incf count))
      (setq ptr (cdr ptr)))
    (setq result (concat result ")"))
    (cons result count)
    ) ;; end LET!
  )   ;; end DEFUN! d-compress-args

(defun d-get-class-list ()
  (interactive)
  (save-excursion
    (save-match-data
      (let (indent-str class-or-interface class-name p1 p2 list)
        (goto-char (point-min)) ;;           1
        (while (re-search-forward (concat "\\(^[ \t]*\\)"
                                          "\\(public[ \t]+\\|abstract[ \t]+\\|"
                                          "final[ \t]+\\|\\)*"
```

```
                                        "\\(class\\|interface\\) +"
                                        "\\([A-Z][a-zA-Z0-9_]*\\)" ) nil t)
            ;;                                3                      4
            (setq indent-str         (buffer-substring-no-properties (match-beginning 1)
                                                                     (match-end 1)))
            (setq class-or-interface (buffer-substring-no-properties (match-beginning 3)
                                                                     (match-end 3)))
            (setq class-name         (buffer-substring-no-properties (match-beginning 4)
                                                                     (match-end 4)))
            (save-excursion
              (beginning-of-line)
              (setq p1 (point))
              (cond
               ((save-excursion
                  (forward-line 1)
                  (beginning-of-line)
                  (looking-at "^[ \t]*{" ))
                (forward-line)
                (beginning-of-line)
                (forward-sexp)
                ;;(if (string= class-name "Singleton")
                ;;    (d-debug "Public Enemy / Mind Terrorist"))
                (setq p2 (point)))
               ((save-excursion
                  (forward-line 1)
                  (beginning-of-line)
                  (looking-at "^[ \t]*begin\\>" ))
                (re-search-forward (concat "^" indent-str "end[ \t]*$" ) nil t)
                (setq p2 (point)))
               )
              (setq list (cons (list class-or-interface class-name p1 p2) list))))
          list))))

(defun d-are-we-inside-class (class)
  (d-assert (stringp (nth 0 class)))
  (d-assert (stringp (nth 1 class)))
  (and (>= (point) (nth 2 class))
       (<= (point) (nth 3 class))))

(defun d-find-matching-class (class-list)
  (block nil
    (let ((ptr class-list)) ;; (setq ptr class-list)
      (while ptr
        (when (d-are-we-inside-class (car ptr))
          ;;(message "* found d-are-we-inside-class class-list=%s (car ptr)=%s" ptr (car ptr))
          ;;(d-error "Foomatic")
          (return (car ptr)))
        (setq ptr (cdr ptr))))))

(defun d-get-enclosing-class ()
  (let (class-list)
    (setq class-list (d-find-matching-class (d-get-class-list)))
    ;;(d-error "Alien Syndrome / class-list=%s" class-list)
    class-list))

;; (setq compress-args (d-compress-args '("100" "200" "300")))
(d-defmacro
 singleton_design_pattern
 (let (ctor compress-args compressed-args compressed-count
            list-of-classes matching-class count location)
   (setq class (nth 1 (d-get-enclosing-class)))
   (d-error (and "Public Enemy / How to Kill a Radio Consultant" class))
   (with-temp-buffer
     ;;(when (get-buffer "*singleton*")
```

```
;;  (kill-buffer "*singleton*"))
;;(switch-to-buffer (generate-new-buffer "*singleton*"))
(setq b2 (current-buffer))
;;(message "* rest=%s" rest)
(setq ctor (nth 0 rest))
(insert ctor)
(goto-char (point-min))
(while (re-search-forward "^\\([ \t]*\\)constructor[ \t]*(" nil t)
  (replace-match (concat "\\1constructor " class "(") 'fixedcase))
(goto-char (point-min))
(d-assert (flm-re-search-forward-no-comments-no-strings "(" nil t))
(setq begy (point))
(setq compress-args (d-compress-args (cdr rest)))
(setq compressed-args (car compress-args))
(setq compressed-count (cdr compress-args))
(setq location (flm-re-search-forward-no-comments-no-strings "(" nil t))
(forward-char -1)
(forward-sexp)
(setq endy (point))
(goto-char begy)
(setq count 0)
(condition-case err
    (while (<= (point) endy)
      (cond
       ;; ------------------------------------------------------
       ((looking-at "[a-zA-Z0-9_]")
        (skip-chars-forward "a-zA-Z0-9_")
        ;;(message "* [a-zA-Z0-9_] (point)=%s line=(%s) count=%s"
        ;;  (point) (d-current-line-as-string) count)
        )
       ;; ------------------------------------------------------
       ((looking-at "[ \t\r\n]")
        (skip-chars-forward " \t\r\n")
        ;;(message "skip-chars-forward  \\t\\r\\n (point)=%s" (point))
        ;;(d-debug "Public Enemy / Don't Believe the Hype")
        )
       ;; ------------------------------------------------------
       ((looking-at ",")
        (incf count)
        ;;(message "* (point)=%s line=(%s) incf count=%s" (point)
        ;;  (d-current-line-as-string) count)
        (forward-char)
        ;;(d-debug "Cold Lampin' with Flavor")
        )
       ;; ------------------------------------------------------
       ((looking-at "/\\*")
        (forward-sexp))
       ;; ------------------------------------------------------
       ((looking-at "\"")
        ;;(error "* inside string")
        (forward-sexp))
       ;; ------------------------------------------------------
       ((looking-at "//")
        (forward-line)
        (beginning-of-line))
       ;; ------------------------------------------------------
       ((looking-at "(")
        (forward-sexp))
       ;; ------------------------------------------------------
       ((looking-at ")")
        (forward-char)
        )
```

```
            ;; -----------------------------------------------------
            ((looking-at "<")
             (forward-sexp))
            ;; -----------------------------------------------------
            ((looking-at "{")
             (let ((debug-on-error nil))
               (error "{ found in arg list")))
            ;; -----------------------------------------------------
            (t
             (message "Misc case (point)=%s" (point))
             (forward-char))))
        (error
         (message "Error err=%s" (prin1-to-string err))))
      (incf count) ;; NOTE: one more than the number of commas
      (let ((debug-on-error nil))
        (when (/= count compressed-count)
          (d-debug "(/= count compressed-count): count=%s compressed-count=%s" count compressed-count)))
      ;;(d-debug "Public Enemy / Raise the Roof (point)=%s" (point))
      (setq ctor (buffer-substring-no-properties (point-min) (point-max)))
      (setq str (concat "private " ctor
                        "private classVar " class " private_instance;"
                        "public function " class " getInstance()"
                        "{"
                        "if (private_instance != null) then "
                        "{"
                        "return private_instance;"
                        "}"
                        "else"
                        "{"
                        "return private_instance = new " class compressed-args ";"
                        "}"
                        "}"))
      ;;(message "str=%s" str)
      str
      ) ;; end WITH-TEMP-BUFFER!
    )   ;; end LET!
  )     ;; end D-DEFMARO! singleton_design_pattern

(defun split-string-into-csv (str)
 "Note: csv stands for Comma Separated Values"
  (with-temp-buffer
    ;;(when (get-buffer "*csv*")
    ;;  (kill-buffer "*csv*"))
                                        ;(set-buffer (generate-new-buffer "*csv*"))
    ;;(switch-to-buffer (current-buffer))
    (setq b3 (current-buffer))
    ;;(switch-to-buffer b3)
    (d-assert (stringp str))
    (insert str)
    (jtw-mode)
    ;;(d-debug "Public Enemy / Public Enemy No. 1")
    ;;(let ((debug-on-error nil))
    ;;  (error "Prince / Forever in my life"))
    (let ((done  nil)
          (endy  nil)
          (p0    (goto-char (1+ (point-min))))
          (p1    nil)
```

```
    (list  nil)
    (depth 0))
(while (not endy)
  (while (not done)
    (message "* schmu depth=%s looking-at=\"%s\""
             depth
             (buffer-substring-no-properties (point) (jtw--clamp-point (+ (point) 10))))
    (condition-case err
        (cond
         ((looking-at "{")
          (condition-case err
              (forward-sexp)
            (error
             (forward-char)
             (incf depth))))
         ((looking-at ",")
          (forward-char 1)
          (when (= depth 0)
            (setq done t)))
         ((looking-at "<")
          (condition-case err
              (progn
                (forward-sexp)
                (cond
                 ((save-excursion
                    (backward-char)
                    (looking-at ">"))
                  ;; DO NOTHING!
                  )
                 ((save-excursion
                    (backward-char)
                    (looking-at ")"))
                  (decf depth)
                  )))
            (error
             (forward-char)
             (incf depth))))
         ((looking-at "[a-zA-Z0-9_]+")
          (skip-chars-forward "a-zA-Z0-9_"))
         ((looking-at "[ \t\r\n]+")
          (skip-chars-forward " \t\r\n"))
         ((eobp)
          (setq done t)
          (setq endy t))
         ((and (looking-at ")") (> depth 0))
          (decf depth)
          (when (= depth 0)
            (setq done t)
            (setq endy t)
            ))
         ((looking-at "(")
          (condition-case err
              (forward-sexp)
            (error
             (forward-char)
             (incf depth))))
         ((looking-at "[")
          (condition-case err
              (forward-sexp)
            (error
             (forward-char 1)
             (incf depth))))
```

```
                ((looking-at "\\]")
                 (forward-char)
                 (decf depth))
                ((looking-at "//")
                 (forward-sexp))
                ((looking-at "/\\*")
                 (forward-sexp))
                ((looking-at "\"")
                 (forward-sexp))
                (t
                 (forward-char)
                 ))
              (error
               ;;(message "Error err=%s" (prin1-to-string err))
               (cond
                ((eq (car err) 'invalid-regexp)
                 ;;(d-debug "invalid-regexp %s" (prin1-to-string err))
                 (forward-char)
                 (setq done t))
                ((eq (car err) 'end-of-buffer)
                 ;;(d-debug "end-of-buffer %s" (prin1-to-string err))
                 (setq done t)
                 (setq endy t))
                ((eq (car err) 'scan-error)
                 (let ((debug-on-error nil))
                   (error "scan error %s" (prin1-to-string err)))
                 (setq done t)
                 (setq endy t))
                (t
                 (let ((debug-on-error nil))
                   (error "Misc error: %s" err)))
               ))))
        (setq done nil)
        (setq p1 (point))
        (setq str (buffer-substring-no-properties p0 (1- p1)))
        (setq p0 p1)
        ;;(d-debug "foomatic")
        ;;(d-assert (null list))
        (setq list (cons str list))
        ;;(sit-and-message 3 "list=%s" list)
        )
      ;;(d-debug "Prince / It's Gonna Be a beautiful night")
      (setq list (nreverse list)))))

(defun splat-list (list)
  ;;(setq args (eval args))
  (let ((done   nil)
        (i      0)
        (result nil))
    (while (not done)
      (if (nth i list)
          (setq result (cons (nth i list) result))
        (setq done t))
      (incf i)
      )
    (setq list (mapcar (function (lambda (x) `(quote ,x))) list))
    list))

(defun fcall (func &rest args)
  (eval `(,func ,@args))
  )

(d-defmacro
 foreach
 (setq vrbl (nth 0 rest))
```

```
 (setq list (nth 1 rest))
 (message "vrbl=%s" vrbl)
 (message "list=%s" list)
 (d-assert (null (cdddr rest)))
 ;;(d-assert (null (nth 3 rest)))
 (concat "for (Iterator " vrbl "= " list ".getIterator(); "
         "!" vrbl ".isDone(); "
          vrbl ".next())" )
 )

(d-defmacro
 null_macro
 (message "(nth 0 rest)=%s" (nth 0 rest))
 (concat "public property String s = " (prin1-to-string (nth 0 rest)) ";" ))

(provide 'd-defmacro)
;; END FILE:    ~/dlisp/d-defmacro.el
```

Study the following fragment of `jtw-build-java.el` (see 2.13.1) which deals with macros:

```
;; BEGIN FILE: el/d-defmacro.el
(progn
  (setq ptr d-macro-list)
  (while ptr
    (while (re-search-forward (prin1-to-string (car ptr)) nil t)
      (when (not (warn-inside-comment-or-string))
        (beginning-of-line)
        (setq p0 (point))
        (skip-chars-forward "a-zA-Z0-9_ \t\r\n" )
        (setq p1 (point))
        (if (not (looking-at "(" ))
            (let ((debug-on-error nil))
              (error "*** Not looking at \"(\" expression" )))
        (forward-sexp 1)
        (setq p2 (point))
        (setq str (buffer-substring-no-properties p1 p2))
        (delete-region p0 p2)
        (setq args (split-string-into-csv str))
        (insert (eval `(fcall (car ptr) ,@ (splat-list args))))
        ))
    (setq ptr (cdr ptr))))
;; END FILE:    el/d-defmacro.el
```

Here is some J.T.W. code that uses the `getter` and `setter` macros:

```
// BEGIN FILE: jtw-tutorials/Foo.jtw
class Foo
begin
    getter (int,foo)
    setter (int,foo)
    getter (int,bar)
    setter (int,bar)
end
// END FILE:    jtw-tutorials/Foo.jtw
```

Here is the resulting Java code:

```
// BEGIN FILE: jtw-tutorials/Foo.java
class Foo
{
   public int getFoo() { return private_foo; }
   public void setFoo(int foo) { private_foo = foo; }
   private int private_foo;

   public int getBar() { return private_bar; }
   public void setBar(int bar) { private_bar = bar; }
   private int private_bar;
}
// END FILE: jtw-tutorials/Foo.java
```

Note that the properties `private_foo` and `private_bar` are automatically created when you call one of `getter` or `setter` macros. This is not the case for the Lisp++ version of the `getter` and `setter` macros.

```
(class X private property int i; private property int j; singleton_design_pattern(constructor
(int i, int j, /* rest of args */)
{ this.i = i; this.j = j; /* rest of ctor code */},100,200,/* rests of ctor parameters */)
)
```

which generates the following Java code:

```
class X
{
   private property int i;
   private property int j;
   private X(int i, int j)
   {
      this.i = i;
      this.j = j;
   }
   private X private_instance;
   public static X getInstance()
   {
      if (private_instance != null)
      {
         return private_instance;
      }
      else
      {
         return private_instance = new X(100,200);
      }
   }
}
```

The `foreach` macro is called like so:

```
// BEGIN FILE: jtw-tutorials/IteratorTest.jtw

class Node
begin
```

```
   property Object current;
   property Node   next;

   constructor Node(Object current)
   begin
      this.current = current;
   end
end

interface Iterator
begin
   public method Iterator first ();
   public method void     next ();
   public method boolean  isDone ();
   public method Object   currentItem ();
end

class SinglyLinkedListIterator implements Iterator
begin
   property Node first;
   property Node current;

   constructor SinglyLinkedListIterator(Node first)
   begin
      this.first   = first;
      this.current = first;
   end

   public method SinglyLinkedListIterator first ()
   begin
      return new SinglyLinkedListIterator(first);
   end

   public method void next ()
   begin
      if (current != null) then
      begin
         current = current.next;
      end
   end

   public method boolean isDone ()
   begin
      return current == null;
   end

   public method Object currentItem ()
   begin
      return current.current;
   end
end

class SinglyLinkedList
begin
   property Node first;

   public method Iterator getIterator ()
   begin
      return new SinglyLinkedListIterator(first);
   end

   public method void addElement (Object o)
   begin
      var Node n = new Node(o);
      n.next = first;
```

```
      first  = n;
   end
end

class IteratorTest
begin
   beginMain
      System.out.println "Welcome to IteratorTest" );
      var SinglyLinkedList list = new SinglyLinkedList();
      list.addElement(123);
      list.addElement(456);
      list.addElement(789);
      list.addElement "apple" );
      list.addElement "banana" );
      list.addElement "carrot" );
      var int i = 0;
      foreach (n,list)
      begin
         System.out.println "i=" + i + ", " + n.currentItem());
         i++;
      end
      System.out.println();
   endMain
end
// END FILE:    jtw-tutorials/IteratorTest.jtw
```

The above code results in the following print out:

```
Welcome to IteratorTest
i=0, carrot
i=1, banana
i=2, apple
i=3, 789
i=4, 456
i=5, 123
```

2.7.2 Proof of concept #2: A superfor macro

One application of the Java preprocessor is the **superfor** macro, which is an enhanced BASIC-style **for** loop. Here is how to invoke the **superfor** macro in your `*.jtw` file:

```
// BEGIN FILE: jtw-tutorials/SuperFor.jtw
class SuperFor
begin
   beginMain
      System.out.println "Welcome to SuperFor.jtw" )
      superfor (var int i = 0 to 10)
      begin
         System.out.println "i=" + i);
      end
   endMain
end
// END FILE:    jtw-tutorials/SuperFor.jtw
```

The above code results in the following printout:

```
Welcome to SuperFor.jtw
i=0
```

```
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
```

The step size argument is optional, here is an example with an explicit step size announced:

```
// BEGIN FILE: jtw-tutorials/SuperFor2.jtw
class SuperFor2
begin
   beginMain
      System.out.println "Welcome to SuperFor2.jtw" )
      superfor (var int i = 0 to 10 step 2)
      begin
         System.out.println "i=" + i);
      end
   endMain
end
// END FILE:    jtw-tutorials/SuperFor2.jtw
```

The above code results in the following printout:

```
Welcome to SuperFor2.jtw
i=0
i=2
i=4
i=6
i=8
i=10
```

If the **downto** keyword is given instead of the **to** keywords then the loop will count downwards from the first given number to the second, even if a positive **step** size is given. Here is an example with a negative step size:

```
// BEGIN FILE: jtw-tutorials/SuperFor3.jtw
class SuperFor3
begin
   beginMain
      System.out.println "Welcome to SuperFor3.jtw" )
      superfor (var int i = 10 downto 0 step 2)
      begin
         System.out.println "i=" + i);
      end
   endMain
end
// END FILE:    jtw-tutorials/SuperFor3.jtw
```

The above code results in the following printout:

```
Welcome to SuperFor3.jtw
i=10
i=8
i=6
i=4
i=2
i=0
```

Note that the specification of the **superfor** macro doesn't need constants as the values of `start`, `stop` and `step-size`. They can be any variable or more generally any Java expression, and those expressions will be evaluated only once, should your code have side effects, i.e. changes the value of a variable in your code. In the following code, the expression `++x` has the side effect of incrementing the value of `x` before returning the value of `x`. Similarly for `fooVariable`. See the following code:

```
// BEGIN FILE: jtw-tutorials/SuperFor4.jtw
class SuperFor4
begin
   classVar int fooVariable = 22;

   function int foo ()
   begin
      return ++fooVariable;
   end

   function int bar ()
   begin
      return 2;
   end

   beginMain
      System.out.println "Welcome to SuperFor4.jtw" )
      var int x = 15;
      superfor (var int i = foo() - bar() to (2 * ++x))
      begin
         System.out.println "i=" + i);
      end
   endMain
end
// END FILE:    jtw-tutorials/SuperFor4.jtw
```

The above code results in the following printout:

```
Welcome to SuperFor4.jtw
i=21
i=22
i=23
i=24
i=25
i=26
i=27
i=28
i=29
i=30
i=31
i=32
```

Elisp source code for the superfor macro

The following code belongs in the file `jtw-build-java.el` which in itself is too large for inclusion in this book (2,900+ lines of code). You can find this code by visiting the following Website:

davin.50webs.com/J.T.W/tutorial-01-HelloWorld.html

and clicking on the tarball in Question 1.1. Alternatively, you can study this fragment of the file `jtw-build-java.el` which deals with the **superfor** macro.

```
;; BEGIN FILE: el/superfor.el
(let (p1 p2 str form type variable T var start stop
        step-size step-size-2 this_start this_stop this_step
        this_step_size file line p-prior beg0 end0
        (case-fold-search nil) from to step keyword-to
        keyword-step-size)
  (setq strobe nil)
  (checkpoint "2")
  (save-excursion
    (goto-char (point-min))
    (setq *superfor* 0)
    (while (re-search-forward "\\<superfor\\>" nil t)
      (checkpoint "found superfor...")
      (setq beg0 (match-beginning 0))
      (setq end0 (match-end 0))
      ;;(checkpoint "sitting for 1 seconds...")
      (font-lock-fontify-buffer)
      (when (save-excursion
              (save-match-data
                (re-search-forward "(" (point-at-eol) t)
                (forward-char -1)
                (re-search-forward "\\<var\\>" (point-at-eol) t)
                (not (warn-inside-comment-or-string))))
        ;;superfor (var int i = 0 to 10)
        ;;(error "Smelly cat")
        (setq *current-buffer* (current-buffer))
        (setq p1 beg0)
        (skip-chars-forward " \t\r\n")
        (when (not
               (save-match-data
                 (looking-at "{")))
          ;; EVAL HERE! vvv

          (setq p2 ;; EVAL HERE! nnn
                (save-excursion
                  (forward-sexp 1)
                  (point)))
          (setq str (buffer-substring-no-properties end0 p2))
          (checkpoint "str=%s" str)
          (setq form (read-str str))
          (checkpoint "form=%s" form)
          ;;(d-debug "form")
          ;;(d-assert (consp form))
          (message "*** form=%s" form)
          ;;(setq debug-on-error nil)
          ;;(error "The Rolling Stones / Rolling Stones plays Cuba")
          (message "(deleted-region=%s)" (buffer-substring-no-properties p1 p2))
          (delete-region p1 p2)
          (incf *superfor*)
          (setq this (format "superfor_%d_" *superfor*))
          (when (not (eq (nth 0 form) 'var))
            (warn-log-message "Error 35: Keyword var missing from superfor construct")
```

```
  )
(when (eq (nth 0 form) 'var)
  (if (and (not (eq (nth 1 form) 'char))
           (not (eq (nth 1 form) 'short))
           (not (eq (nth 1 form) 'int))
           (not (eq (nth 1 form) 'long))
           (not (eq (nth 1 form) 'float))
           (not (eq (nth 1 form) 'double)))
      (warn-log-message (concat
                         "Error 37:#2 argument type to superfor macro must be"
                         " one of char/short/int/long/float/double")))
  ;; (setq form '(var int i=0 to stop))
  ;; (setq form '(var int i =0 to stop))
  ;; (setq form '(var int i = 0 to stop))
  (progn
    (setq form-str (aref (eval `(d-prin1-to-string-java ,form sexy)) 0))
    (when (string-match "^var[ \t]*" form-str)
      (setq form-str (substring form-str (match-end 0))))
    (when (string-match "^\\(char\\|short\\|int\\|long\\|float\\|double\\)\\>" form-str)
      (setq T                    (substring form-str (match-beginning 0) (match-end 0)))
      (setq form-str (d-trim-string (substring form-str (match-end 0))))
      (when (string-match "[^<>]=" form-str)
        (setq var (substring form-str 0 (1+ (match-beginning 0))))
        (setq form-str (substring form-str (1+ (length var))))
        ))
    (cond
     ((string-match "\\<to\\>" form-str)
      (message "found to")
      (setq keyword-to 'to)
      (setq start      (d-trim-string (substring form-str 0 (match-beginning 0))))
      (setq form-str   (d-trim-string (substring form-str (match-end 0))))
      )
     ((string-match "\\<downto\\>" form-str)
      (message "found downto")
      (setq keyword-to 'downto)
      (setq start    (d-trim-string (substring form-str 0 (match-beginning 0))))
      (setq form-str (d-trim-string (substring form-str (match-end 0))))
      )
     ) ;; END COND!
    )
  ;;(d-debug "Duran Duran / Girls on Film")
  ;;(setq form '(var int i = 0 to 10 step 2))
  (progn
    (if (string-match "\\<step\\>" form-str)
        (progn
          (setq keyword-step-size t)
          (setq stop (d-trim-string (substring form-str 0 (match-beginning 0))))
          (setq step (d-trim-string (substring form-str (match-end 0))))
          )
      (setq keyword-step-size nil)
      (setq stop (d-trim-string form-str))
      (setq step nil)
      )
    ;;(setq start form)
    ;;(when (string-match "=" start)
    ;;  (setq start (substring start (match-end 0))))
    ;;(when (string-match "\\<to\\>" start)
    ;;  (setq start (d-trim-string (substring start 0 (match-beginning 0)))))
    ;;(setq rest1 (eval `(d-prin1-to-string-java , form step)))
    ;;(setq stop (aref rest1 0))
    ;;(when (string-match "\\<to\\>" stop)
    ;;  (setq stop (d-trim-string (substring stop (match-end 0)))))
    ;;(setq keyword-step (car (aref rest1 1)))
```

```
  ;;(when keyword-step
  ;;  (setq step (aref rest1 1))
  ;;  (when (eq keyword-step 'step)
  ;;    (setq step (cadr (aref rest1 1)))
  ;;    (if step (setq keyword-step-size 'step))))
  )           ;; END PROGN!
;;(d-debug "Art Blakey / Lou's Blues")
(progn ;; (warn--cull-quotes)
  ;;(setq var         eq)
  (setq start-2     (warn-splat-quest start))
  (setq stop-2      (warn-splat-quest stop ))
  (setq step-size-2 (warn-splat-quest step ))
  ) ;; END PROGN!
;; ----------------------------------------------
;;(d-debug "The Pretenders / Precious")
(setq this_start     (concat this "start" ))
(setq this_stop      (concat this "stop" ))
(setq this_step      (concat this "step" ))
(setq this_step_size (concat this "step_size" ))
;;(d-debug "Dire Straits / My Parties")
(insert (concat (concat "var " T " " this_start " = " start-2 "; " )
                (concat "var " T " " this_stop " = " stop-2 "; " )
                (if keyword-step-size
                    (concat "var " T " " this_step " = " step-size-2 "; "
                            "var " T " " this_step_size " = "
                             (cond
                              ((eq keyword-to 'to)
                               (concat "Math.abs(" this_step ")" ))
                              ((eq keyword-to 'downto)
                               (concat "-Math.abs(" this_step ")" ))
                              (t
                               (d-debug "Dire Straits / Heavy Fuel" )))
                            ";\n"
                             )
                  (concat "var " T " " this_step_size " = "
                           (cond
                            ((eq keyword-to 'to)
                            "1" )
                            ((eq keyword-to 'downto)
                            "-1" )
                            (t
                             (d-debug "Dire Straits / Ticket to Heaven" )))
                           ";\n"
                           ) ;; END CONCAT!
                  )          ;; END IF!
                )            ;; END CONCAT!
          )                  ;; END INSERT!
  )                          ;; END PROGN!
;;(d-debug "Rod Stewart / Hot Legs")
(setq line 0)
(setq p-prior
      (save-excursion
        (beginning-of-line)
        (setq str (concat "^[ \t]*//+ " *pp-namespace* "#location[0-9]"
                          " (\\(" *drive-spec* "[-a-zA-Z0-9_./]+\\):\\([0-9]+\\))" ))
        (if (or (looking-at str) (re-search-backward str nil t))
            (progn
              ;;(d-debug "Antonio Vivaldi")
              (setq file (buffer-substring-no-properties          (match-beginning 1)
```

```
                                                                    (match-end 1)))
                    (d-assert (stringp file))
                    (setq line (read-str (buffer-substring-no-properties (match-beginning 3)
                                                                         (match-end 3))))
                    (d-assert (integerp line))
                    (point)
                    )
                  (setq file (concat *def-dir* *stump* ".jtw"))
                  (setq line 1)
                  (goto-char (point-min))
                  (forward-line 2)
                  (point)
                  )))
          (setq line (+ line (count-lines p-prior (point))))
          (decl line)
          (decl line)
          (insert (format "// %s '%s\n"                    *list-namespace* (prin1-to-string file-stack)))
          (insert (format "// %s#location3 (%s:%d)\n"      *pp-namespace*   file line))
          (insert (concat "for (var " T " " var " = " this_start ";"
                          " ((" this_step_size " > 0) ? " var " <= "
                          this_stop " : " var " >= " this_stop "); "
                          var " += " this_step_size ")" ))
          (if strobe (d-debug "Pretenders / The Wait"))
          ;;(d-debug "Yehudi Menuhin")
          ) ;; END WHEN!
        )   ;; END WHEN!
      )     ;; END WHILE!
    )       ;; END SAVE-EXCURSION!
  )         ;; END LET!
```

;; END FILE: el/superfor.el

A bug in J.T.W. superfor

The question mark **operator** `a ? b : c` which expands to

```
type result;
if (a) then
begin
   result = b;
end
else
begin
   result = c;
end
```

where `type` can be any Java type directly supported by the arguments to the **superfor** macro in J.T.W., namely char, short, int, long, float and double. Elsewhere the question mark is supported. Instead in the **superfor** macro you have to write the following code to get a question mark **operator** online:

```
// BEGIN FILE: jtw-tutorials/SuperFor5.jtw
class SuperFor5
begin
```

```
    beginMain
       System.out.println "Welcome to SuperFor5.jtw" );
       foo(1,2);
    endMain
    function void foo (int x, int y)
    begin
       superfor (var int i=0 to (x < y) QUEST 10 : 20))
       begin
          System.out.println "i=" + i);
       end
       System.out.println();
    end
end

// END FILE:    jtw-tutorials/SuperFor5.jtw
```

where the symbol QUIST compiles into a question mark: ? When built, the program prints out the following:

```
Welcome to SuperFor5.jtw
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
```

2.7.3 Proof of concept #3: File inclusion

When your classes become large and unwieldy, it becomes useful to split a source file into several compilation units. The most natural division into compilation units is at the level of **method**s. With each method in a separate file you can manage **method**s that are excessively large. Here is how to use file inclusion in the J.T.W. language. First comes the `*.jtw` file with all bodies of **method**s harvested from them:

```
class Foo

   include STRINGBGFG("apple.method")
   include STRINGBGFG("banana.method")
   include STRINGBGFG("carrot.method")
```

Here are the files that get included. The first file is `apple.method`:

```
property int prop; /* property for use with apple method */

method void apple (/* parameters */)

   prop = prop + 1;
   /* rest of body of apple method */
```

The second file is `banana.method`:

```
method void banana (/* parameters */)

   /* body of banana method */
```

The third file is `carrot.method`:

```
method void carrot (/* parameters */)

   /* body of carrot method */
```

When all of the file inclusions have been carried out by the J.T.W. to Java compiler, the code that `javac` sees will be something like this:

```
/** Automatically generated file. Do not edit! */
// #foomatic #location (Foo.jtw:1)
class Foo
{
   // #foomatic #location (apple.method:1)
   int prop;

   void apple (/* parameters */)
   {
      prop = prop + 1;
      /* rest of body of apple method */
   }
   // #foomatic #location (banana.method:1)
   void banana (/* parameters */)
   {
      /* body of banana method */
   }
   // #foomatic #location (carrot.method:1)
   void carrot (/* parameters */)
   {
      /* body of carrot method */
   }
   // #foomatic #location (Foo.jtw:6)
}
```

Note the use of the value `#foomatic` of the string `*pp-namespace*` (where *pp* stands for pre-processor) that is a long arbitrarily defined string to prevent accidental aliasing with the rest of the commented code that the user of the system might write. The `#location` directives are used to keep track of the original line number in the source file. Using Emacs batch mode executing the Elisp code: `jtw-build-java.el` (see 2.13.1), error messages in `Foo.java` now point back to the original `Foo.jtw` file, or one of the files that get **#include**d like so: `apple.method`, `banana.method` or `carrot.method`.

NOTE: Version 1.0 of J.T.W. used the C Pre-Processor (C.P.P.) to manage the `#location` directives but unfortunately C.P.P. destroys comments in the target file, and Java uses `/** ... */` comments to document the program's behaviour so C.P.P. cannot be used.

2.8 Java/J.T.W./C++ coding preferences

Many a religious war has been fought over coding preferences, how code should be named and indented. I started programming when I was 5 years old in 1978 so over my 40 years as a computer programmer I have gravitated to the following coding preferences. Here I present them to you now, and I also explain their rationale so that their use is not mindlessly following my own religious ideas but rather practical conventions for improving the readability of program code. The recommended preferences for indenting J.T.W. code is as follows:

```
begin
   /* code goes here */
   begin
      /* code goes here */
      begin
         /* code goes here */
      end
      /* code goes here */
   end
   /* code goes here */
end
```

In Emacs you can get the above indentation online by putting the following command in your `~/.emacs` file, where ~ is an abbreviation for the contents of your `HOME` environment variable.

```
(setq c-basic-offset 3)
```

instead of:

```
begin begin /* code goes here */ end begin /* code goes here */ end end
```

or similar coding styles. The rationale for placing ends in equal alignment with begins is so that even on long lines, the **begin** and **end** symbol are not truncated away from view, unless you are not looking at column zero, which is a rare event, or you have a pathologically deep level of nesting of your squigglies (curly braces) i.e. more than screen width divided by tab width = 80 / 3 = 26 on my system. Note that in Emacs, `screen-width` is a **function** and `tab-width` is a variable so you can calculate this value in your version of Emacs by evaluating the following code:

```
(/ (screen-width) tab-width).
```

In Emacs activate **Control-x Control-e** at the end of the above Lisp form to execute that code. The only place where this falls down is where you have excessively long lines which are ugly no matter how your editor chooses to display them. In Emacs the variable `truncate-lines` can either be set to `t` in which long lines keep the screen scrolling to the right hand side of the screen. When `nil` the lines wrap around inside the visible window of the screen. Both approaches look ugly in my opinion. Luckily the programmer is able to reformat their code so that excessively long lines do not occur. This coding preference for J.T.W. code translates into the following preference for Java and C/C++ code:

```
{
   /* code goes here */
   {
      /* code goes here */
      {
         /* code goes here */
```

```
        }
        /* code goes here */
    }
    /* code goes here */
}
```

The much maligned *Hungarian Notation* is recommended so that syntax highlighting can be applied to keywords. The term "Hungarian Notation" comes from the fact that under the worst instances of Hungarian notation such as `m_piMax` your code looks as indecipherable as the Hungarian language is to Westerners. In Hungarian notation, **private property**s and **method**s should be named with a preceding underscore like so: `_foo` or something similar like `private_foo`. The famous book *Design Patterns* by [GRHV95] uses an underscore at the beginning of a word to indicate that that variable is **private**. The following Elisp code can allow **private property**s to be highlighted in a different color from the rest of your code:

```
;; BEGIN FILE: ~/dlisp/d-flock-private.el
;; END FILE:   ~/dlisp/d-flock-private.el
```

Simply place this code into your file `.emacs` in your `HOME` directory and run Emacs to activate this syntax highlighting feature. If such a file does not exist, it will be necessary to create one.
Java and J.T.W. conventionally name **variable**s in "caMeL" case, i.e. component words concatenated together and using uppercase letters to delimit the sub-words of a given expression. Examples are like so: `setFoo()` and `getFoo()`. In C and C++ symbols are conventionally named with underscores like so: `set_foo()` and `get_foo()`. If you follow these conventions, your code will be easier to read by the large number of other programmers who follow these conventions.

2.9 Parenthesis and squigglies { ... } instead of begin ... end

It is sometimes said that Lisp stands for *Lots of Irritating Superfluous Parentheses*. But in reality Lisp is for the expert coder who prefers their programming to be deeply nested. In the same vein, going from BASIC to Java involves getting used to squigglies { ... } all over the place. The Basic coder will soon find that { ... } operators are a useful tool for managing the complexity of a program. While learning a program language for the first time however, the programmer will like as much help as the compiler can give you, which includes supporting the **begin** and **end** constructs.

2.10 Troubleshooting J.T.W. code

The Elisp file `jtw-build-java.el` (see 2.13.1) contains code for GNU Emacs to parse and troubleshoot problematic J.T.W. code. The following errors produce a diagnostic:

- Error 1: **method** needs a return type.
- Error 2: **function** needs a return type.
- Error 3: **constructor**s need the correct **class** name.
- Errors 5-13: Cannot have more than one of **property**, **classVar**, **function**, **method** or **constructor** on the same line.
- Error 14: This line needs one of the following keywords: **function**, **method**, **classVar**, **property** or **constructor**.

- Error 15: Functions cannot reside inside **function**s/**method**s/**constructor**s.
- Error 16: Function must have **begin** on the following line.
- Error 17: Constructors cannot reside inside **function**s/**method**s/**constructor**s.
- Error 18: **constructor** must have **begin** on the following line.
- Error 19: Methods cannot reside inside **function**s/**method**s/**constructor**s.
- Error 20: Method must have **begin** on the following line.
- Error 21: Property must not have **begin** on the following line.
- Error 22: Class variable must not have **begin** on the following line.
- Error 23: Expecting (after **if** statement.
- Error 24: Unbalanced parentheses after **if** statement.
- Error 25: Expecting **then** keyword after **if** statement.
- Error 26: More **end**s than **begin**s.
- Error 27: Missing **end**s at the end of the file.
- Error 28: Spurious semicolon at the end of the line.
- Error 29: Cannot call a **method** without an object from the **main** **function**.
- Error 30: Cannot call a **method** with a **class** name prefix from the **main** **function**.
- Error 31: Cannot call a **method** without an object from a **function**.
- Error 32: Cannot call a **method** with a **class** name prefix from a **function**.
- Error 33: Cannot call a **method** without an object from a **method**.
- Error 34: Cannot call a **method** without an object from a **constructor**.
- Error 35: Keyword **var** missing.
- Error 36: Keyword **var** does not belong here.
- Error 37: argument type to **superfor** macro must be one of char/short/int/long/float/double.
- Error 38: **function** outside of a **class**.
- Error 39: **method** outside of a **class**.
- Error 40: **property** outside of a **class**.
- Error 41: Class variable outside of a **class**.
- Error 42: Cannot have a **function** inside an **interface**.
- Error 44: Class `X` has no **function** named `foo`.
- Error 45: Class `X` has no **classVar** named `foo`.
- Error 46: Function `Foo.bar()` not found.
- Error 47: ClassVar `Foo.classVar` not found.
- Error 48: Infinite loop in include directives.
- Error 49: **class** `X` has multiple instances.

2.11 Mapping from J.T.W. to Java

The J.T.W. language maps to the Java language in a natural and straightforward way, making it easy to learn Java, once you know the J.T.W.language. Here is the actual mapping of keywords from J.T.W. to Java:

```
function    → static
var         → nothing
classVar    → static
property    → nothing
method      → nothing
constructor → nothing
begin       → {
end         → }
beginMain   → public static void main (String args) {
endMain     → }
and         → &&
or          → ||
then        → nothing
elseif      → else if
```

2.11.1 Choosing a preprocessor language for J.T.W.

Note that these J.T.W. keywords on the left hand side of the above diagram should not map to their Java equivalents inside strings and comments. The transformation was originally written to use the m4 language to map J.T.W. onto Java but this approach had the disadvantage that keywords like **begin** and **end** inside strings were mapped to their Java equivalents like so:

```
System.out.println(STRINGBGFG("function")); → System.out.println(STRINGBGFG("static"));
System.out.println(STRINGBGFG("var")); → System.out.println(STRINGBGFG(""));
System.out.println(STRINGBGFG("classVar")); → System.out.println(STRINGBGFG("static"));
System.out.println(STRINGBGFG("property")); → System.out.println(STRINGBGFG(""));
System.out.println(STRINGBGFG("method")); → System.out.println(STRINGBGFG(""));
System.out.println(STRINGBGFG("constructor")); → System.out.println(STRINGBGFG(""));
System.out.println(STRINGBGFG("begin")); → System.out.println(STRINGBGFG("{"));
System.out.println(STRINGBGFG("end")); → System.out.println(STRINGBGFG("}"));
System.out.println(STRINGBGFG("beginMain")); → System.out.println(STRINGBGFG("public static
void main(String[] args) {"));
System.out.println(STRINGBGFG("endMain")); → System.out.println(STRINGBGFG("}"));
System.out.println(STRINGBGFG("and")); → System.out.println(STRINGBGFG("&&"));
System.out.println(STRINGBGFG("or")); → System.out.println(STRINGBGFG("||"));
System.out.println(STRINGBGFG("then")); → System.out.println(STRINGBGFG(""));
System.out.println(STRINGBGFG("elseif")); → System.out.println(STRINGBGFG("else if"));
```

which is of course the wrong behaviour. A hack to get around this limitation is to break apart the J.T.W. keywords like so:

```
System.out.println(STRINGBGFG("be") + STRINGBGFG("gin"));
```

This problem can be fixed for good either by using *Flex* to compile J.T.W. into Java or to use Emacs to do the same thing, only a little slower than what Flex can do. In the end I chose GNU Emacs as the host for the preprocessor language J.T.W. because it is free software and is adequate for my programming needs and is more powerful than Flex or m4. To remedy this deficiency Emacs' batch mode is used to do the transformation from J.T.W. to Java. This implies that GNU Emacs must be present on the client's system to do the J.T.W. to Java mapping. Of course, there

is no compulsion to use Emacs as an editor, although there are a couple of advantages in doing this. Number one is that J.T.W. keywords, comments and strings have syntax **highlighting**. And number two is that Emacs can do correct automatic indentation of J.T.W. code.

2.11.2 Piping the output of `javac` and `java`

Output from the executables `javac` and `java` have their standard output stream and error stream piped into Emacs' batch mode so that error messages like `Foo.java:123` point back to the correct file even if file inclusion (see §2.7.3) has been used. The programs `grep` and `sed` are also used as pipes in the transformation process so they must be present on the client's system.

2.11.3 The *GNU Makefile* for building *.java files and `*.class` files

Here is the `Makefile` that is used to build ***.java** files from `*.jtw` files and `*.class` files from ***.java** files and finally executing `*.class` files:

```
.PRECIOUS:
.PRECIOUS: %.java %.class

JAVAC_FLAGS = -source 1.5 -Xlint:unchecked -Xlint:deprecation -Xlint:-options
JAVA_FLAGS = -enableassertions
SHELL = /bin/bash
PREFIX = /usr/
TELEPHONE = telephone-1800-NEW-FUNK

build-class-db:
        @echo STRINGBGFG("* Stage 0 :  Building class database")
        emacs --batch --eval STRINGBGFG("(setq dir \"$(PREFIX)/share/emacs/site-lisp/dlisp/\")") \
--load $(PREFIX)/share/emacs/site-lisp/dlisp/jtw-build-class-db.el --funcall doit

%.java : %.jtw
        @echo STRINGBGFG("* Stage 1 :  Debugging $*.jtw and building $*.java file") \
        emacs --batch --eval STRINGBGFG("(setq *stump* \"$*\")") \
--load $(PREFIX)/share/emacs/site-lisp/dlisp/jtw-build-java.el \
--funcall doit

%.class: %.java
        @echo STRINGBGFG("* Stage 2 :  Debugging *.java file(s) and building *.class file(s)")
        javac $(JAVAC_FLAGS) $$(find .  -name STRINGBGFG("*.java")) |& emacs --batch \
--load $(PREFIX)/share/emacs/site-lisp/dlisp/jtw-javac.el --funcall doit |& \
grep STRINGBGFG("#$(TELEPHONE) input[0-9]:") - |& sed -e STRINGBGFG("s/\#$(TELEPHONE) input[0-9]://g") -

%.run: %.class
        @echo STRINGBGFG("* Stage 3 :  Running $*.class file")
        java $(JAVA_FLAGS) $* |& emacs --batch \
--load $(PREFIX)/share/emacs/site-lisp/dlisp/jtw-java.el --funcall doit \
|& grep STRINGBGFG("#$(TELEPHONE) input[0-9]*:") - |& sed -e STRINGBGFG("s/\#$(TELEPHONE)
input[0-9]*://g") -

clean: build-class-db
        rm -fv $$(find .  -name STRINGBGFG("*.java"))
        rm -fv $$(find .  -name STRINGBGFG("*.class"))

build: clean
```

The first line **.PRECIOUS** without any arguments clears the list of precious files, the list of files not to delete during the build process.

2.12 Elisp code for editing `*.jtw` files

This following Elisp file `$(PREFIX)/share/emacs/site-lisp/dlisp/jtw-mode.el` gives you syntax **highlighting** of J.T.W. constructs and correct indentation of J.T.W. code.

```
;; BEGIN FILE: ~/dlisp/d-make-face.el

;; (d-make-face 'red-face (setq bgcolor bg-colour) "red" :bold)
(defmacro d-make-face (font bgcolor fgcolor &rest rest)
  ;;(d-debug "Queen / Another one bites the dust")
  (d-assert (symbolp 'font))
  (d-assert (if (boundp 'font)
                (symbolp 'font)
              t))
  ;;(d-debug "Calamansi")
  (let (p was-error
        bold unbold
        italic unitalic
        underline ununderline)
    ;;(d-debug "The Shape of Jazz to Come / Chronology")
    ;;(d-debug "Queen / Fat Bottomed Girls")
    (setq bgcolor (eval bgcolor))
    (setq fgcolor (eval fgcolor))
    ;;(message "bgcolor=%s fgcolor=%s" bgcolor fgcolor)
    ;;(progn (setq bgcolor "#ffffff") (setq fgcolor "#000") (setq font 'fg:white))
    (setq p `(progn
               (if (not (eq 'font 'default))
                   (kill-local-variable (quote , font)))
               (setq , font (quote , font))
               (make-face (quote , font))
               (set-face-background (quote , font) , bgcolor)
               (set-face-foreground (quote , font) , fgcolor)))
    (setq ptr rest)
    ;;(d-debug "The Shape of Jazz to Come / Congeniality")
    (while ptr
      (cond
       ((or (null    (car ptr))
            (stringp (car ptr)))
        )
       ;; ------------------------------------------------------------
       ((or (eq (car ptr) :bold) (eq (car ptr) :unbold))
        (if (eq (car ptr) :bold)
            (setq bold t))
        (if (eq (car ptr) :unbold)
            (setq unbold t))
        (when (and bold unbold)
          (setq was-error (concat
                           was-error
                           "Both symbols should not be defined: :bold and :unbold," )))
        (if bold
            (setq p `(progn
                       , p
                         (make-face-bold (quote , font)))))
        (if unbold
            (setq p `(progn
                       , p
                         (make-face-unbold (quote , font))))
          ))
       ;; ------------------------------------------------------------
       ((or (eq (car ptr) :italic) (eq (car ptr) :unitalic))
        (if (eq (car ptr) :italic)
            (setq italic t))
        (if (eq (car ptr) :unitalic)
            (setq unitalic t))
        (when (and italic unitalic)
          (setq was-error (concat
```

```
                        was-error
                        "Both symbols should not be defined: :italic and :unitalic," )))
        (if italic
            (setq p `(progn
                       , p
                         (make-face-italic (quote , font)))))
        (if unitalic
            (setq p `(progn
                       , p
                         (make-face-unitalic (quote , font))))
          ))
        ;; ------------------------------------------------------------
        ((or (eq (car ptr) :underline) (eq (car ptr) :ununderline))
         (when (eq (car ptr) :underline)
           (setq u-or-uu   t)
           (setq underline t))
         (when (eq (car ptr) :ununderline)
           (setq u-or-uu nil)
           (setq ununderline t))
         (when (and underline ununderline)
           (setq was-error (concat
                        was-error
                        "Both symbols should not be defined: :underline and :ununderline," )))
         (setq p `(progn
                    , p
                      (set-face-underline (quote , font) u-or-uu))))
        ;; ------------------------------------------------------------
        (t ;; (setq was-error "Schmu")
         ;;(d-debug "Calamansi")
         (if (not (car ptr))
             (debug))
         (setq was-error (format "%s, FOO! unrecognised symbol: %s"
                                 was-error
                                 (car ptr)))
         (error (format "%s Unrecognised keyword %s" was-error (car ptr))))
        )
       (setq ptr (cdr ptr))) ;; end WHILE! ptr
      ;; ------------------------------------------------------------
      (if was-error
          (d-error (concat was-error " in macro d-make-face." ))
        )
      p)
    )

;; (d-amiga-color (setq rgb-components "#fff"))
(defun d-amiga-color (rgb-components)
  "Allows for entry into the Amiga colour-space with 12 bits of
  colour for a total of 4096 different colours."
  (cond
   ((= (length rgb-components) 7)
    rgb-components)
   ((= (length rgb-components) 4)
    (let (r g b)
      (setq r (substring rgb-components 1 2))
      (setq g (substring rgb-components 2 3))
      (setq b (substring rgb-components 3 4))
      (setq rgb-components (concat "#" r r g g b b))
      ))))

(progn
  (setq bg-colour "#f0f0f0" )
  (setq prefs-bg-black-p nil)
  (setq bg-colour-inverted "#000" )
  )
```

```
(defun d-font-lock-add-begin (keywords)
  (if (fboundp 'font-lock-add-keywords)
      (font-lock-add-keywords nil keywords nil)
    (setq font-lock-keywords
          (append
           keywords
           font-lock-keywords))))

(defun d-font-lock-add-end (keywords)
  (if (fboundp 'font-lock-add-keywords)
      (font-lock-add-keywords nil keywords 'end)
    (setq font-lock-keywords
          (append
           font-lock-keywords
           keywords))))

(provide 'd-make-face)

(quote
 abc)

(d-quote
 abc
 def)
;; END FILE:     ~/dlisp/d-make-face.el


;; BEGIN FILE: ~/dlisp/jtw-mode.el
;;; jtw-mode.el — A new major mode for editing *.jtw files

;; Copyright (C) 2016 Davin Pearson

;; Maintainer: Davin Max Pearson <http://davin.50webs.com>
;; Keywords: Java Training Wheels major mode
;; Version: 2.0

;;; Commentary:

;; This program is part of GNU Java Training Wheels.

;;; m4_limitation_of_warranty

;;; Code:

(require 'cl)

;;(setq d-emergency-set-load-path--dir (format "%s/dlisp/" (getenv "PWD")))

;;(when (not (fboundp 'd-emergency-set-load-path))
;;  (defun d-emergency-set-load-path ()
;;    (setq load-path (cons d-emergency-set-load-path--dir load-path))
;;    (message "jtw-mode.el (car load-path)=%s" (car load-path))
;;    ))
;; (d-emergency-set-load-path)

(require 'early-bindings)

(defvar jtw-mode-syntax-table)

(defvar jtw-mode-map (make-keymap))

(setq auto-mode-alist (cons '"\\.jtw$" . jtw-mode) auto-mode-alist))

(add-hook 'font-lock-mode-hook 'd-jtw-font-lock-mode-hook-post 'APPEND)

(defun cull-from-list (cull-me list)
  (let (ptr)
```

```
    (setq ptr list)
    (while ptr
      (when (equal cull-me (car ptr))
        (setq list (cdr ptr))
        (setq ptr  nil)
        )
      (setq ptr (cdr ptr)))
    list))

(defun d-jtw-font-lock-mode-hook-post ()
  (if (eq major-mode 'jtw-mode)
      (d-font-lock-add-end
       '(
         "^[ \t]*\\(//.*$\\)" 1 'font-lock-comment-face t))))

(defvar *elaborate-jtw* t
 "Whether or not to turn on buggy java-mode syntax highlighting")

(defun jtw-mode ()
  (interactive)
  ;;(html-mode)
  ;;(if *elaborate-jtw*
  (java-mode)
  (setq major-mode 'jtw-mode)
  (setq mode-name "JTW")
  (set (make-local-variable 'jtw-mode-syntax-table)
       (copy-syntax-table java-mode-syntax-table))
  (set-syntax-table jtw-mode-syntax-table)
  (progn
    (modify-syntax-entry ?_ "w")
    (modify-syntax-entry ?< "(>")
    (modify-syntax-entry ?> ")<")
    )
  (use-local-map jtw-mode-map)
  (local-set-key "\t"                 'jtw-indent-line)
  (progn
    (local-set-key "\C-m"               'd-indent-new-comment-line)
    (local-set-key "\C-r"               'd-indent-new-comment-line)
    )
  (local-set-key [(meta control \\)] 'jtw-meta-control-backslash)
  (local-set-key "\C-c\C-c"           'd-cc-comment-region)
  (abbrev-mode 1)
  (setq local-abbrev-table java-mode-abbrev-table)
  (make-local-variable 'font-lock-keywords)
  (make-local-variable 'c-basic-offset)
  (setq c-basic-offset 3)
  (font-lock-mode 1)
  (font-lock-fontify-buffer)
  ;;(setq font-lock-keywords nil)
  ;;; NOTE: the following code adds syntax highlighting of /** ... */ javadoc comments
  (when *elaborate-jtw*
    (setq font-lock-keywords (cull-from-list
                              '("\\<\\(@[a-zA-Z0-9]+\\)\\>" (1 c-annotation-face))
                              font-lock-keywords))
    (set (kill-local-variable 'global-font-lock-keywords) font-lock-keywords)
    (with-temp-buffer
      (emacs-lisp-mode)
      (kill-local-variable 'global-font-lock-keywords)
      (insert-prin1 '(setq global-font-lock-keywords
                           (append global-font-lock-keywords
                                   '(c-font-lock-complex-decl-prepare
                                     (#[(limit)
                                        sexy-string
```

```
                                        [limit javadoc-font-lock-doc-comments c-font-lock-doc-comments "/\\*\\*" ]
                                        4])))))
    (goto-char (point-min))
    (d-assert (re-search-forward "\\<sexy-string\\>" nil t))
    (replace-match (format "\"\302\303%c%c#\207\"" 8 ?\t) 'FIXEDCASE 'LITERAL)
    (eval-buffer)
    (setq font-lock-keywords global-font-lock-keywords)
    ))
;; NOTE: the following code adds fontication of J.T.W. keywords
(when *elaborate-jtw*
  (d-font-lock-add-begin
   '(
     "\\(class\\) \\([A-Z][a-zA-Z0-9_]*\\)"
      (1 'font-lock-keyword-face nil)
      (2 'font-lock-type-face    t))

     (,(concat "\\<\\([A-Z]+[a-z][A-Za-z0-9]*\\|[A-Z]\\|void\\|boolean\\|"
               "char\\|int\\|long\\|short\\|float\\|double\\)"
               "[][]*[ \t]+\\([a-z][A-Za-z0-9_]*\\)(")
      (1 'font-lock-type-face          nil)
      (2 'font-lock-function-name-face nil))

     (,(concat "\\<\\([A-Z]+[a-z][A-Za-z0-9]*\\|[A-Z]\\|void\\|boolean\\|"
               "char\\|int\\|long\\|short\\|float\\|double\\)"
               "[][]*[ \t]+\\([a-z][A-Za-z0-9_]*\\) *[;=,)]")
      (1 'font-lock-type-face          nil)
      (2 'font-lock-variable-name-face nil))

     (,(concat "\\<\\(d-assert\\|function\\|var\\|classVar\\|"
               "property\\|method\\|constructor\\|"
               "until\\|then\\|and\\|or\\|include\\)\\>")
      (1 font-lock-keyword-face nil))

     "^\\(package\\)[ \t]+\\([a-z.]+\\);"
      (1 'bold nil)
      (2 'fg:lightred t))

     "^\\(import\\)[ \t]+\\([a-z.]+\\)\\.\\*;"
      (1 'bold nil)
      (2 'fg:lightred t))

     "\\<\\(begin\\)\\>"                          0 font-lock-keyword-face nil)
     "\\<\\(end\\)\\>"                            0 font-lock-keyword-face nil)
     "\\<\\(beginMain\\)\\>"                      0 font-lock-keyword-face nil)
     "\\<\\(endMain\\)\\>"                        0 font-lock-keyword-face nil)

     "\\<\\(System.out.print\\(ln\\)?\\)("        1 d-face-cc-global nil)
     "\\<\\(System.exit\\)("                      1 d-face-cc-global nil)
     "\\<\\([a-z][A-Za-z0-9]*\\.printStackTrace\\)(" 1 d-face-cc-global nil)
     "\\<\\(null\\|true\\|false\\)\\>"            1 font-lock-constant-face nil)

     (,(concat "\\<\\(abstract\\|break\\|byte\\|case\\|catch\\|"
               "const\\|continue\\|default\\|do\\|else\\|elseif\\|"
               "extends\\|final\\|finally\\|for\\|goto\\|if\\|"
               "implements\\|instanceof\\|interface\\|"
```

```
                "native\\|new\\|package\\|private\\|protected\\|"
                "public\\|return\\|static\\|super\\|switch\\|"
                "synchronized\\|this\\|throw\\|throws\\|transient\\|"
                "superfor\\|downto\\|to\\|step\\|"
                "try\\|volatile\\|while\\)\\>" )
          1 font-lock-keyword-face nil)

         "\\(\\<\\|-\\)\\([0-9]+[.]\\)?[0-9]+\\([eE]-?[0-9]+\\)?"
         0 d-face-cc-digits nil)

         "\\<b\\>" 0 'bold nil)

         "\\<function [^ \t]* \\([a-z][A-Za-z0-9_]*\\)("
         1 font-lock-function-name-face nil)
         "\\<method [^ \t]* \\([a-z][A-Za-z0-9_]*\\)("
         1 font-lock-function-name-face nil)

         "\\<\\(method\\|function\\) \\([a-z][a-zA-Z0-9_]*\\)("
         2 font-lock-function-name-face nil)

         "\\<[A-Z]+[a-z][A-Za-z0-9_<,>]*"    0 'font-lock-type-face nil)
         "\\<[A-Z]\\>"                       0 'font-lock-type-face nil)
         (,(concat "\\<\\(void\\|boolean\\|char\\|int\\|long\\|short\\|"
                   "float\\|double\\)\\>" )   0 'font-lock-type-face nil t)

         "\\<m4_[a-zA-Z0-9]*"                0 d-face-m4 t)

         (,(concat "\\(\\<m4_" "dnl\\>\\)\\([^\r\n]*\\)$" )
          (1 d-face-m4-dnl t)
          (2 font-lock-comment-face t))
         "\\<\\(\\([a-z]+\\.\\)*\\)[A-Z][a-zA-Z0-9_]*" 1 'fg:lightred nil)

         "\\<\\(getter\\|setter\\)\\>"                                1 'd-face-defmacro t)
         "\\<\\([a-zA-Z0-9_]*[\\|]?_design[\\|]?_pattern\\)\\>" 1 'd-face-defmacro t)
         "\\<\\(foreach\\)\\>"                                        1 'd-face-defmacro t)

        )))
  ;;(font-lock-fontify-buffer)
  )

(defun jtw--clamp-point (newpoint)
  (max (point-min) (min (point-max) newpoint)))

(defun jtw--inside-comment-or-string ()
  (save-match-data
    (let ((p (get-char-property (jtw--clamp-point (1- (point))) 'face)))
      (or (eq p 'font-lock-string-face)
          (eq p 'font-lock-comment-face)
          (eq p 'font-lock-doc-face)
          (eq p 'font-lock-doc-string-face)
          (eq p 'd-face-super-comment)
          )))
  )

(defun jtw--count-string (string)
  (save-excursion
    (save-match-data
      (let ((max (point-at-eol))
            (count 0))
        (beginning-of-line)
```

```
        (while (re-search-forward string max t)
          (if (not (jtw--inside-comment-or-string))
              (incf count)))
        count))))

(defun jtw--count ()
  (let (r)
    (save-excursion
      (beginning-of-line)
      (setq r (- (+ (jtw--count-string "\\<begin\\>")
                    (jtw--count-string "\\<beginMain\\>")
                    (* 2 (jtw--count-string "("))
                    (* 2 (jtw--count-string "{")))
                 (+ (jtw--count-string "\\<end\\>")
                    (jtw--count-string "\\<endMain\\>")
                    (* 2 (jtw--count-string ")"))
                    (* 2 (jtw--count-string "}")))))
      ;;(message "r=%s" r)
      r)))

(defun jtw--get-indent ()
  (save-excursion
    (beginning-of-line)
    (while (looking-at " ")
      (forward-char))
    (- (point) (point-at-bol))))

(defun jtw--set-indent (should-be)
  (if (>= should-be 0)
      (save-excursion
        (beginning-of-line)
        (d-assert (looking-at "^[ \t]*"))
        (setq i (- (match-end 0) (match-beginning 0)))
        (when (/= i should-be)
          ;;(d-foo)
          (delete-region (point-at-bol)
                         (save-excursion
                           (beginning-of-line)
                           (skip-chars-forward " ") (point)))
          (beginning-of-line)
          (insert (make-string should-be ? ))))))

(defvar jtw--basic-offset 3)

(defun jtw--line-1 ()
  (interactive)
  ;;(d-foo)
  (save-excursion
    (beginning-of-line)
    ;;(d-foo)
    (cond
     ((= (point) (point-min))
      ;;(d-foo)
      (jtw--set-indent 0))
     ((looking-at "^[a-z ]*\\(class\\|interface\\)\\>")
      (when (not (flm-inside-comment-or-string))
        (jtw--set-indent 0)))
     (t
      (forward-line -1)
      (setq rel (jtw--count))
      (setq i (jtw--get-indent))
```

```
      (forward-line 1)
      ;;(if (/= rel 0) (beep))
      ;;(set-buffer-modified-p t))
      (jtw--set-indent (+ i (* rel jtw--basic-offset)))))))

(defun jtw--line-2 ()
  ;;(d-foo)
  (save-excursion
    (when (looking-at "^[ \t]*end")
      (setq i (jtw--get-indent))
      (jtw--set-indent (- i jtw--basic-offset)))))

;;(eval '(setq f 123))
;;(setq func 'jtw--line-1)
;;(eval (cons 'jtw--line-1 nil))

(defun jtw--a (func)
  (save-excursion
    (let (m)
      (setq m (make-marker))
      (forward-line)
      (set-marker m (point))
      (if (not (re-search-backward "^\\([a-z].*\\)?\\(class\\|interface\\)" nil t))
          (goto-char (point-min)))
      ;;(d-foo)
      ;;(goto-char (point-min))
      (while (< (point) (marker-position m))
        (eval (cons func nil))
        (forward-line 1))
      (set-marker m nil))))

(defun jtw--meta-control-backslash ()
  (interactive)
  (let (m)
    (setq m (make-marker))
    (set-marker m (point))
    (if (and (fboundp 'd-movement--unpad-buffer) (d-movement--is-correct-mode))
        (d-movement--unpad-buffer))
    (goto-char (point-min))
    (while (< (point) (point-max))
      (jtw--line-1)
      (forward-line 1))
    (goto-char (point-min))
    (while (< (point) (point-max))
      (jtw--line-2)
      (forward-line 1))
    (if (and (fboundp 'd-movement--pad-buffer) (d-movement--is-correct-mode))
        (d-movement--pad-buffer))
    (goto-char m)
    (set-marker m nil)
    (message "Ran jtw--meta-control-backslash")
    ))

(defun jtw--all ()
  ;;(d-beeps "line1")
  (jtw--a 'jtw--line-1)
  ;;(d-beeps "line2")
  (jtw--a 'jtw--line-2)
  ;;(d-beeps "line3")
  )

(defun jtw--get-indents ()
  (save-excursion
    (let (list)
      (goto-char (point-max))
      (beginning-of-line)
```

```
      (setq list nil)
      (while (not (bobp))
        (forward-line -1)
        (beginning-of-line)
        (setq i (jtw--get-indent))
        (setq list (cons i list)))
      list)))

(defun jtw--newline ()
  (interactive)
  (let (c)
    (when (save-excursion (beginning-of-line) (looking-at "^.*//"))
      (setq c t))
    ;;(d-foo)
    (insert "\n")
    (jtw--indent-line)
    (if c (insert "// "))))

(defun jtw--delete-line ()
  (delete-region (point-at-bol) (point-at-eol))
  (if (looking-at "\n")
      (delete-char 1))
  )

(defun jtw--get-current-indentation ()
  (save-excursion
    (beginning-of-line)
    (d-assert (looking-at "^\\([ \t]*\\)[^ \t\r\n]"))
    (/ (length (buffer-substring-no-properties (match-beginning 1) (match-end 1)))
       c-basic-offset)))

(defun jtw--current-line-as-string ()
  (buffer-substring-no-properties (point-at-bol)
                                  (point-at-eol)))

(defun jtw--get-prev-and-this-line ()
  (beginning-of-line)
  (let (line)
    (list (if (save-excursion
                (beginning-of-line)
                (bobp))
              ""
            (save-excursion
              (forward-line -1)
              (beginning-of-line)
              (while (and (not (bobp)) (looking-at "^[ \t]*$"))
                (forward-line -1)
                (beginning-of-line))
              (setq line (d-what-line))
              ;;(message "*** jtw--current-line-as-string=%s" (jtw--current-line-as-string))
              (jtw--current-line-as-string)))
          (jtw--current-line-as-string)
          line)))

(defun jtw--indent-line ()
  (interactive)
  (font-lock-fontify-buffer)
  (let (pair prev-line this-line i triple)
    (save-match-data
      (save-excursion
        (beginning-of-line)
        (setq i (if (save-excursion
                      (beginning-of-line)
                      (bobp))
```

```
                    0
                   (save-excursion
                     (forward-line -1)
                     (beginning-of-line)
                     (while (and (not (bobp)) (looking-at "^[ \t]*$" ))
                       (forward-line -1)
                       (beginning-of-line))
                     (jtw-get-current-indentation)
                     ;;(debug "bar")
                     )))
              (setq triple (jtw-get-prev-and-this-line))
              ;;(debug "John Coltrane")
              (setq prev-line (nth 0 triple))
              (setq this-line (nth 1 triple))
              (setq previous-nontrivial-line (nth 2 triple))
              (if (and (string-match "begin" prev-line)
                       (save-excursion
                         (goto-line previous-nontrivial-line)
                         (or (looking-at "^[ \t]*begin" )
                             (re-search-forward "begin" (point-at-eol) t)))
                       (not (memq (cadr (text-properties-at (save-excursion
                                                              (goto-line previous-nontrivial-line)
                                                              (beginning-of-line)
                                                              (re-search-forward "begin" (point-at-eol) t))))
                                  '(font-lock-string-face
                                    font-lock-comment-face
                                    font-lock-doc-face
                                    font-lock-doc-string-face
                                    d-face-super-comment))))
                  (incf i))
              (if (and (string-match "end" this-line)
                       (save-excursion
                         (beginning-of-line)
                         (or (looking-at "^[ \t]*end" )
                             (re-search-forward "end" (point-at-eol) t)))
                       (not (memq (cadr (text-properties-at (save-excursion
                                                              (beginning-of-line)
                                                              (re-search-forward "end" (point-at-eol) t))))
                                  '(font-lock-string-face
                                    font-lock-comment-face
                                    font-lock-doc-string-face
                                    font-lock-doc-face
                                    d-face-super-comment))))
                  (decf i))
              (setq i (max 0 i))
              ;;(message "indenting line %d to %d" (d-what-line) i)
              ;;(sit-for 1)
              (beginning-of-line)
              ;;(indent-line-to i)
              (indent-line-to (* c-basic-offset i))
              ;;(debug "Halloway")
              )
            (beginning-of-line)
            (skip-chars-forward " \t" )
            ;;(debug "antelope")
            )))

(require 'd-make-face)

;; I am a normal comment
;;; I am a super comment

(defvar jtw-mode-patch-colors t
  "Set me to nil to prevent overwriting of default colorisation."
```

```
  )

;; ordinary comment
;;; super comment

(setq bg-colour "#f0f0f0")

(require 'd-make-face)

(provide 'jtw-mode)
;; END FILE:    ~/dlisp/jtw-mode.el
```

2.13 Translator `*.jtw` to `*.class` Elisp source code

2.13.1 `jtw-build-java.el` Elisp source code

The file `jtw-build-java.el` saves to disk a ***.java** file corresponding to the `*.jtw` file given as an argument. It gives error diagnostics on problematic J.T.W. constructs. This file respects file line numbers in the case that **include** statements are present in your code. The large size of the file (2,900+ lines of code) makes it unsuitable for inclusion in this book, so instead for the Elisp source code, see the file `jtw-build-java.el` by visiting the following Website:

davin.50webs.com/J.T.W/tutorial-01-HelloWorld.html

and clicking on the tarball in Question 1.1. If you use the default setting of the installer module, the file `jtw-build-java.el` will be located at `/usr/share/emacs/site-lisp/dlisp/` for GNU/Linux and `c:/java-training-wheels/share/emacs/site-lisp/dlisp/` for M.S. Windows.

2.13.2 `jtw-javac.el` Elisp source code

The file `jtw-javac.el` is used to convert ***.java** files to `*.class`, again respecting line numbers in the case that **include** statements are present in your source code. The location of `jtw-javac.el` will be the same as the location of `jtw-build-java.el`. The output of the `javac` command has its standard output and standard error piped into Emacs' batch mode running the file `jtw-javac` and invoking the **method**: `doit`. Here is the file `jtw-javac.el`. This file is included in the tarball mentioned in the last subsection §2.13.1.

```
;; BEGIN FILE:  ~/dlisp/jtw-javac.el

;;; jtw-javac.el --- A program for receiving the output of the program: javac

;; Copyright (C) 2006-2016 Davin Pearson

;; Author/Maintainer: Davin Max Pearson <http://davin.50webs.com>
;; Keywords: javac backend
;; Version: 2.0

;; This program is part of GNU Java Training Wheels.

;;; m4_limitation_of_warranty

;;; Commentary:

;; A program for receiving the output of the program: javac in the form
;; of a pipe.

;;; Known Bugs:

;; None so far!
```

;;; Code:

```
(require 'cl)

(setq *prefix*
      (let ((pwd (getenv "PWD"))) ;; (setq pwd (getenv "PWD"))
        (if (not noninteractive)
            (if (not (string-match "dlisp" pwd))
                (concat pwd "/preprocessors/jtw-projects/")
              pwd)
          pwd)))

(defun checkpoint (msg &rest rest)
  (apply 'message msg rest)
  ;; do nothing
  )

(if (not (boundp 'file-comes-from))
    (setq file-comes-from nil))

(if (not file-comes-from)
    (setq file-comes-from (cons "jtw-javac.el" file-comes-from)))

(setq load-path (cons (expand-file-name (concat *prefix* "/../dlisp"))
                      load-path))

(require 'early-bindings)
(require 'jtw-build-java)

(message "*** Welcome to file: jtw-javac.el %s %s"
         (print-symbol *prefix*)
         (print-symbol *stump*)
         )

(d-assert (find "jtw-javac.el" file-comes-from :test 'string=))
;;(d-assert (string= file-comes-from "jtw-javac.el"))

;;(message "*** Symbol value... %s" (print-symbol *stump*))

(defun doit ()
  (interactive)
  ;;(read-line-pre)
  ;;(message "input8: jtw-javac: *stump*=%s" *stump*)
  (message "*** Called defun: doit file: jtw-javac.el %s"
           (print-symbol *stump*))
  (let (numb said-message red-line numb file-less-suffix old-suffix new-suffix
             line-left line-right file-plus-suffix location
             (case-fold-search t) p)
    (condition-case err
        ;;(while (setq red-line (d-read-line))
        (while (setq red-line (read-from-minibuffer ""))
          (setq said-message nil)
          ;;(message "input0: red-line=%s" red-line)
          ;;(if (not (string-match "^Loading " red-line))
          (cond
           ((or (string-match (regexp-quote "Loading 00debian-vars...") red-line)
                (string-match (regexp-quote "Loading /etc/emacs/site-start.d/50autoconf.el") red-line)
                (string-match (regexp-quote "Loading /etc/emacs/site-start.d/50dictionaries-common.el") red-line)
                (string-match (regexp-quote "Loading debian-ispell...") red-line)
                (string-match (regexp-quote "Loading /var/cache/dictionaries-common/emacsen-ispell-default.el...") re
                (string-match (regexp-quote "Loading /var/cache/dictionaries-common/emacsen-ispell-dicts.el...") red-
                (string-match (regexp-quote "Loading /etc/emacs/site-start.d/50git-core.el") red-line)
```

```
    )
 ;; do nothing
 )
((string-match (concat "\\(\\([a-zA-Z]:/\\|"
                       "~/\\|/\\|\\.\\.+/\\|\\)"
                       "[a-zA-Z0-9_/]+\\)"
                       "\\(\\.java\\):\\([0-9]+\\)" )
               red-line)
 (progn
   (setq file (substring red-line (match-beginning 0) (match-end 3)))
   ;;(message "input6: filename=%s" file)
   (save-match-data
     (if (string-match "^/" file)
         (setq file (substring file 1))))
   ;;(message "input7: filename=%s" file)
   ;;(setq said-message t)
   (setq numb             (1- (d-read-str (substring red-line
                                                     (match-beginning 4)
                                                     (match-end 4)))))
   (setq file-less-suffix (substring red-line
                                     (match-beginning 1)
                                     (match-end 1)))
   ;;(message "input3: red-line=%s" red-line)
   ;;(message "input3: file-less-suffix=%s" file-less-suffix)
   (setq old-suffix ".java" )
   (setq new-suffix ".jtw" )
   (setq line-left        (substring red-line 0 (match-end 1)))
   (setq line-right       (substring red-line (match-end 4)))
   (setq file-plus-suffix (concat file-less-suffix new-suffix))
   (setq file             (concat file-less-suffix old-suffix))
   (if (string-match "./" file)
       (setq file (substring file (match-end 0))))
   ;;(setq default-directory (file-name-directory default-directory))
   ;;(setq file (concat default-directory file))
   ;;(error "Maria Callas")
   ;;(message "input8: (file-name-directory file)=%s" (file-name-directory file))
   ;;(message "input7: file=%s" file)
   ;;(message "input7: default-directory pre=%s" default-directory)
   (d-assert (stringp file))
   ;;(message "input7: file=%s" file)
   ;;(message "input9: (file-name-directory file)=%s" (file-name-directory file))
   (when (file-name-directory file)
     (d-assert (stringp (file-name-directory file)))
     (d-assert (stringp default-directory))
     (if (string-match (file-name-directory file) default-directory)
         (setq default-directory (substring default-directory 0 (match-beginning 0))))
     ;;(message "input7: default-directory post=%s" default-directory)
     ;;(message "input7: (file-name-nondirectory)=%s" (file-name-nondirectory file))
     )
   (d-assert (stringp file))
   (d-assert (stringp default-directory))
   ;;(message "input8: (concat default-directory file)=%s" (concat default-directory file))
   ;;(message "input8: numb=%s" numb)
   (find-file (concat default-directory file))
   ;;(message "input2: finding file=%s" file)
   ;;(debug "Desolation Row")
   (goto-line numb)
   ;;(debug "Tiger Woods")
   ;;(message "input2: Amber Dempsey")
   ;;(message "input2: (buffer-file-name)=%s" (buffer-file-name))
   (setq location (warn-get-location))
   ;;(message "input2: (cdr location)=%d" (cdr location))
   ;;(message "input2: setq location")
   (setq red-line (concat line-left new-suffix ":" (prin1-to-string (cdr location)) line-right))
```

```
            ;;(message "input2: setq red-line")
            ;;(debug "J.S. Bach / Mass in B Minor")
            (message "%s input1: %s" *java-namespace* red-line)))
          (t
           (message "%s input2: %s" *java-namespace* red-line))))
      (error
       (setq p (prin1-to-string (cdr err)))
       (if (and (not (string-match "Error reading from stdin" p))
                (not (string-match "End of file"               p))
                (not (string-match "Eobp"                      p)))
           (message "%s input4: Error=%s" *java-namespace* (cdr err)))
       )))
  (message "*** end defun: doit file: jtw-javac.el %s" (print-symbol *stump*))
  )

(message "*** Scanner reached end file: jtw-javac.el")
;; (round (/ (d-what-line) 58.0)) 2 pages
(provide 'jtw-javac)
;; END FILE:    ~/dlisp/jtw-javac.el
```

2.13.3 `jtw-java.el` Elisp source code

The file `jtw-java.el` reads the output of `java`'s standard output and standard error piped into this file and generates correct line numbers of `java` error messages, even if file inclusion is used. The location of `jtw-java.el` will be the same as the location of `jtw-build-java.el`. Here is the file `jtw-java.el`. This file in included in the tarball mentioned two subsections ago, in §2.13.1.

```
;; BEGIN FILE:  ~/dlisp/jtw-java.el

;;; jtw-java.el --- A program for receiving the output of the program: java

;; Copyright (C) 2006-2016 Davin Pearson

;; Author/Maintainer: Davin Max Pearson <http://davin.50webs.com>
;; Keywords: java backend
;; Version: 2.0

;; This file is part of GNU Java Training Wheels.

;;; m4_limitation_of_warranty

;;; Commentary:

;; A program for receiving the output of the program: java in the form
;; of a pipe.

;;; Known Bugs:

;; None so far!

;;; Code:

(message "Welcome to jtw-java.el")

(require 'cl)

(when (or (not (boundp '*prefix*)) (not *prefix*))
  (message "Foomatic: *prefix* is not bound"))

(when (or (not (boundp '*prefix*)) (not *prefix*))
  (setq *prefix*                    ;; (setq env (getenv "PWD"))
        (let ((env (getenv "PWD"))) ;; (setq env (getenv "PWD"))
```

```
          (if (not noninteractive)
              (if (not (string-match "dlisp" env))
                  (expand-file-name
                   (concat env "/preprocessors/jtw-projects/"))
                env)
            env))))

(when (or (not (boundp '*prefix*)) (not *prefix*))
  (message "Foomatic: *prefix* is not bound"))

;;(assert *prefix*)

(message "Welcome to jtw-java.el SNIFFLER")

;;(d-assert nil)

(if (not (boundp 'file-comes-from))
    (setq file-comes-from nil))

(setq file-comes-from (cons "jtw-java.el" file-comes-from))

(setq load-path (cons (expand-file-name (concat *prefix* "/../dlisp/"))
                      load-path))

;;; NOTE begin: (require 'early-bindings)
;;;
(require 'early-bindings)
;;;
;;; NOTE end: (require 'early-bindings)

(message "Ride on the Peace Train")

(d-assert (boundp '*prefix*))

(message "load-path=%s" (prin1-to-string load-path))

(message (print-symbol *prefix*))

(require 'jtw-build-java)

(d-assert (find "jtw-java.el" file-comes-from :test 'string=))

(defun checkpoint (msg &rest rest)
  ;;(apply 'message msg rest)
  ;; do nothing
  )

(defun doit ()
  (interactive)
  (message "Welcome to defun: doit file: jtw-java.el DOUGHNUTS")
  (let (red-line said-message numb file-less-suffix old-suffix
                 new-suffix line-left line-right file-plus-suffix
                 cdr-err)
    (condition-case err
        (while (setq red-line (read-from-minibuffer ""))
          ;;(while (setq red-line (d-read-line))
          ;;(message "input0: red-line=%s" red-line)
          ;;(message "1")
          (d-assert red-line)
          ;;(message "2")
          (d-assert (stringp red-line))
          ;;(message "3")
          (d-assert (sequencep red-line))
          ;;(message "4")
          (setq said-message nil)
          ;;(message "5")
```

```
(cond
 ((or
   (string-match (regexp-quote "Loading 00debian-vars..." ) red-line)
   (string-match (regexp-quote "Loading /etc/emacs/site-start.d/50aut(string (regexp-quote oconf.el" ) red-l
   (string-match (regexp-quote "Loading /etc/emacs/site-start.d/50dictionaries-common.el" ) red-line)
   (string-match (regexp-quote "Loading debian-ispell..." ) red-line)
   (string-match (regexp-quote "Loading /var/cache/dictionaries-common/emacsen-ispell-default.el" ) red-line
   (string-match (regexp-quote "Loading /var/cache/dictionaries-common/emacsen-ispell-dicts.el" ) red-line)
   (string-match (regexp-quote "Loading /etc/emacs/site-start.d/50git-core.el" ) red-line)
   )
  ;; do nothing
  )
 ((string-match "\\([A-Z][a-zA-Z0-9_]*\\)\\(\\.java\\):\\([0-9]+\\)" red-line)
  ;;(message "6")
  (setq said-message t)
  ;;(message "7")
  (setq numb (substring red-line (match-beginning 3) (match-end 3)))
  ;;(message "8")
  (d-assert (d-read-ready numb))
  ;;(message "9")
  ;;(d-assert (sequencep (count-locations)))
  ;;(setq numb (- (d-read-str numb) (count-locations)))
  ;;(message "10")
  (d-assert (sequencep numb))
  ;;(message "11")
  (d-assert (stringp numb))
  (setq numb (d-read-str numb))
  ;;(message "12")
  (d-assert (integerp numb))
  ;;(d-assert (sequencep numb))
  ;;(message "13")
  (d-assert (stringp red-line))
  (d-assert (sequencep red-line))
  (d-assert (and 1 (match-beginning 1)))
  (d-assert (and 2 (match-end 1)))
  (d-assert (and 3 (match-beginning 2)))
  (d-assert (and 4 (match-end 2)))
  (d-assert (and 5 (match-beginning 3)))
  (d-assert (and 6 (match-end 3)))
  ;;(message "14")
  (setq file-less-suffix (substring red-line (match-beginning 1) (match-end 1)))
  ;;(message "15")
  (d-assert file-less-suffix)
  (d-assert (stringp file-less-suffix))
  ;;(message "16")
  (setq old-suffix ".java" )
  ;;(message "17")
  (d-assert old-suffix)
  (d-assert (stringp old-suffix))
  ;;(message "18")
  (setq new-suffix ".jtw" )
  ;;(message "19")
  (d-assert new-suffix)
  (d-assert (stringp new-suffix))
  ;;(message "20")
  (setq line-left        (substring red-line 0 (match-beginning 1)))
  (setq line-right       (substring red-line (match-end 3)))
  (setq file-plus-suffix (concat file-less-suffix new-suffix))
  (setq file             (concat file-less-suffix old-suffix))
  ;;(message "21")
  (d-assert (stringp line-left))
  (d-assert (stringp line-right))
  (d-assert (stringp file-plus-suffix))
```

```
                (d-assert (stringp file))
                ;;(message "22")
                (find-file file)
                ;;(message "23")
                (d-assert (integerp numb))
                (goto-line numb)
                ;;(message "(warn--get-location)=%s" (warn--get-location))
                ;;(message "24")
                ;;(debug "Tiger Woods")
                (setq location (warn-get-location))
                ;;(setq location (cons file numb))
                ;;(message "24b")
                ;;(message "location=%s" location)
                (d-assert (not (eq location t)))
                (d-assert (not (eq location nil)))
                (d-assert (sequencep location))
                (d-assert (consp location))
                (d-assert (stringp (car location)))
                (d-assert (numberp (cdr location)))
                ;;(message "25")
                (when location
                  ;;(message "26")
                  (setq red-line (concat line-left (car location) ":" (prin1-to-string (cdr location)) line-right))
                  ;;(message "27")
                  )
                ;;(message "28")
                (d-assert (sequencep red-line))
                )
               )  ;; end COND!
              (when said-message
                (message "%s input1: %s" *java-namespace* red-line))
              (when (not said-message)
                (message "%s input2: %s" *java-namespace* red-line))
              ;;(message "Jean Jarre's Equinoxe")
              )
            (error
             (setq cdr-err (prin1-to-string (cdr err)))
             (if (or (string-match "Error reading from stdin"          cdr-err)
                     (string-match "Eobp"                              cdr-err)
                     (string-match "Could not find or load main class" cdr-err))
                 (message "Known error err=%s" cdr-err)
               (message "%s input3: Unknown error (%s)" *java-namespace* cdr-err)
               ) ;; end IF!
             )   ;; end ERROR!
            )    ;; end CONDITION-CASE! err
          )      ;; end LET! red-line said-message numb file-less-suffix old-suffix
        (message "Reached end of defun: doit file: jtw-java.el DOUGHNUTS")
        )

;; My Fair Lady / Rex Harrison & Julie Andrews
(message "Scanner at end of file: jtw-java.el")

;; (round (/ (d-what-line) 50.0)) 3 pages
(provide 'jtw-java)
;; END FILE:    ~/dlisp/jtw-java.el
```

2.14 An idiom for constructors in Java and C++

When a **constructor**'s purpose is to set one or many **property variable**s, it seems natural to name the parameters with the same names as the **property**s. The problem with this approach is that you need to distinguish between the names of the **property**s with the names of the parameters. Luckily there is a way to do this. The **this** keyword is not learned by novice programmers because it is implicit in every mention of a **property** in the same **class** and every call to a **method** of the same **class**. Here is some J.T.W. code to show you what I mean:

```
class A

   property int data;
   method void foo ()

      System.out.println(STRINGBGFG("data=") + data);
      bar(); PRINTS OUT: bar!

   method void bar ()

      System.out.println(STRINGBGFG("bar!"));
```

The `foo` **method** can be identically rewritten as follows:

```
class A

   property int data;
   method void foo ()

      System.out.println(STRINGBGFG("data=") + this.data);
      this.bar(); PRINTS OUT: bar!

   method void bar ()

      System.out.println(STRINGBGFG("bar!"));
```

Therefore `this.data` inside the A **class** is the same as `data` and `this.bar()` inside the A **class** is the same as `bar()`. A difference occurs when there is a parameter called `data`, in which case `this.data` and `data` refer to different **variable**s, the former to the **property** `data` and the latter to the parameter `data`. You can exploit this difference by writing your **constructor** like so:

```
class A


   property int data;

   constructor A(int data)

      this.data = data;
```

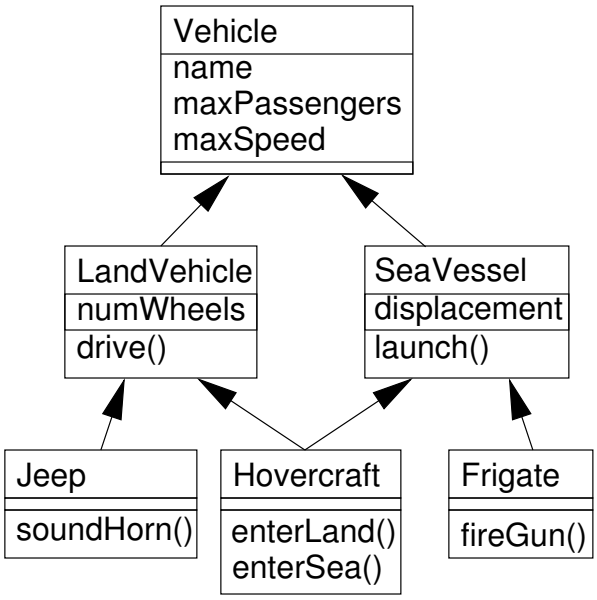


Figure 2.3: A U.M.L diagram for C++

or for more parameters, like so:

```
class A


   property int data1;
   property int data2;
   property int data3;

   constructor A(int data1, int data2, int data3)

      this.data1 = data1;
      this.data2 = data2;
      this.data3 = data3;
```

The only difference between the Java code and C++ code is that **this** in C++ is a pointer to the current object rather than a reference to the current object like it is in Java. Therefore in C++ and Lisp++ you write **this->**`data` rather than **this.**`data` in Java and J.T.W.

2.15 Interfaces in Java and J.T.W.

This section explains how **interface**s in Java and J.T.W. are a solution to C++'s problematic multiple inheritance. Consider Figure 2.3 for an example. The `Hovercraft` **class** shown in the diagram inherits from both `LandVehicle` and `SeaVessel` since the hover-craft is in the rather unique position of being able to travel on land and sea. The `Hovercraft` **class** cannot be expressed

in Java since Java does not have the facility for *multiple inheritance*. All other **class**es in the diagram use *single inheritance* and so they can be expressed in Java.

One of the problems with multiple inheritance is in deciding what to do with **property**s in a **class** like `Vehicle` that is an indirect superclass of `Hovercraft` in two different ways, via `LandVehicle` and via `SeaVessel`. The hover-craft in being able to drive on land and sea might have two different maximum speeds, one for land travel and the other for sea travel. This leads to a problem of what should be the appropriate value for the `maxSpeed` **property** of `Hovercraft` objects? We could set `maxSpeed` to be the maximum of the two speed values but then this might badly affect the behaviour of the `drive` **method** which, because it is defined in the `LandVehicle` **class**, might assume that the value of `maxSpeed` is the maximum speed attainable on land. A similar problem arises with the `launch` **method**.

Another approach would be for the `Hovercraft` **class** to possess two separate `maxSpeed` **property**s, one for the maximum speed on land and the other for the maximum speed on the sea. The C++ language gives the programmer a choice between having one or two copies of `maxSpeed` with the option of using **virtual** base **class**es rather than normal inheritance, whereas Java avoids this extra complexity by not allowing multiple inheritance.

So that the Java programmer is not disadvantaged by the lack of multiple inheritance, Java has the **interface** feature, which allows for a kind of multiple inheritance involving **interface**s, without the complexity of multiple inheritance of **class**es that is present in languages like C++. Figure 2.4 shows on the left a diagram showing how **interface**s in Java relate to the Java concepts of **class**es and objects. On the right is a diagram showing the equivalent concepts in C++.

The diagram shows that in a sense **interface**s are a "higher level concept" than **class**es, since you can never create an instance of an **interface**, only instances of **class**es that implement that **interface**. Interfaces have no **constructor**s.

The most important feature of **interface**s is that a **class** can implement more than one interface. Interfaces are limited in two respects. Firstly, they are not allowed to have any **property**s except **static** constants, and secondly the **method**s of an **interface** must be defined without bodies, like **abstract method**s. These two limitations prevent **interface**s from suffering from the problem that occurred with the `maxSpeed` **property** in the previous U.M.L. diagram.

We can re-work the previous U.M.L. diagram into something that can be expressed within the Java language by replacing the **class**es `Vehicle`, `LandVehicle` and `SeaVessel` with **interface**s `IsVehicle`, `IsLandVehicle` and `IsSeaVessel`, respectively. The dotted arrows in Figure 2.5 indicate **interface**s extending from **interface**s. Note that the `Hovercraft` **class** implements both the `IsLandVehicle` and `IsSeaVessel` **interface**s, rather than inheriting from two **class**es which is not allowed in Java.

Since an **interface** is not allowed to have any **property**s except **static** constants, we have replaced the **property**s that existed in the **class**es `Vehicle`, `LandVehicle` and `SeaVessel` with "getter" and "setter" **method**s. That is to say that, for each **property** `X`, there is now a pair of **method**s `getX` and `setX`. A `getX`, `setX` pair of **public method**s in a **class** is logically equivalent for users of the **class** to a **public property** called `X`. Since the **method**s of the **interface**s are defined without bodies, they are defined in the **class**es `Jeep`, `Hovercraft` and `Frigate` that implement the three **interface**s. The `getMaxSpeed()` **method** could return the maximum speed depending on whether or not the vehicle is currently on the land or on the sea, and similarly for the `setMaxSpeed()` **method**.

2.16 Packages in Java and J.T.W.

2.16.1 Package visibility

In Java and J.T.W. when an object is declared with package visibility it gains a level of protection between **protected** and **private**.

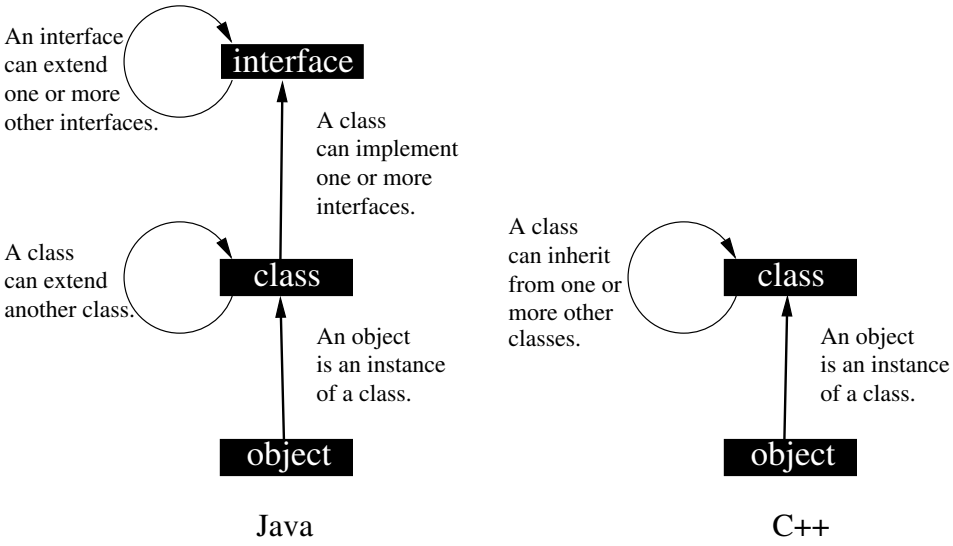


Figure 2.4: Comparision of Java's objects, **class**es and **interface**s with C++'s objects and **class**es. Note that to simulate Java's **interface**s in C++ it is sufficient to use **abstract class**es, that is to say: **class**es with at least one pure **virtual method**.

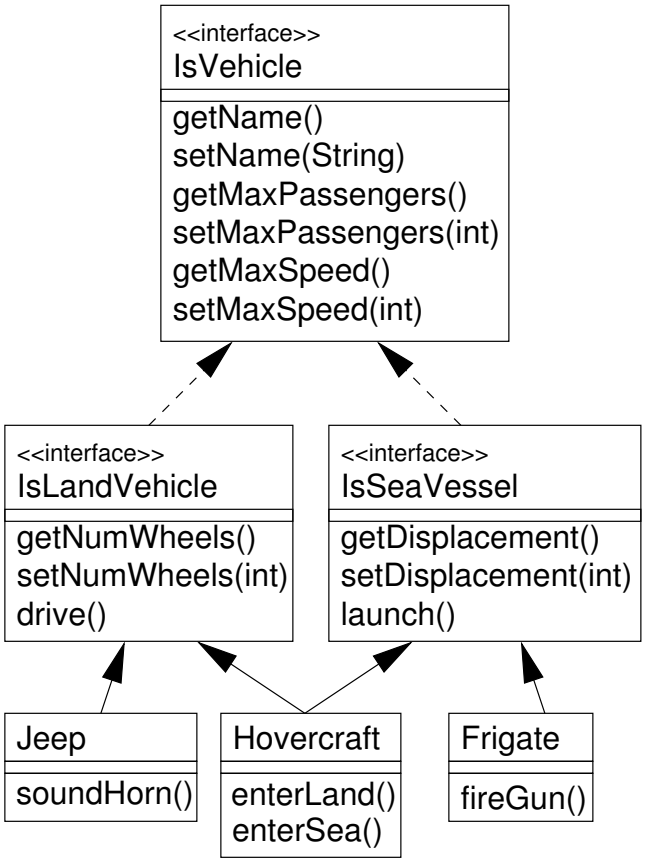


Figure 2.5: A U.M.L diagram for Java. Note that dotted lines represent **interface**s extending from one another.

	public visibility	**protected** visibility	**package** visibility	**private** visibility
In the same **class** as `X`	✔	✔	✔	✔
In the same **package** as `X`	✔	✔	✔	✖
In a subclass of `X` but a different **package**	✔	✔	✖	✖
Anywhere else	✔	✖	✖	✖

To get **package** visibility, simply omit **public**, **private** and **protected** from the **method**, **property** or **constructor** spec, e.g. like so in J.T.W.:

```
// BEGIN FILE: A.jtw
class A
begin
   function void package_visible_function()
   begin
      // NOTE: code goes here
   end
   method void package_visible_method()
   begin
      // NOTE: code goes here
   end
   property int package_visible_property;

   classVar int package_visible_class_variable;
end
// END FILE:    A.jtw
```

and like so in Java:

```
// BEGIN FILE: A.java
class A
{
   static void package_visible_function()
   {
      // NOTE: code goes here
   }
   void package_visible_method()
   {
      // NOTE: code goes here
   }
   int package_visible_property;

   static int package_visible_class_variable;
}
// END FILE:    A.java
```

2.16.2 Moving a class into a package

Consider a typical **class**:

```
// BEGIN FILE: jtw-tutorials/A.jtw
class A
begin
   property int data;

   classVar int data2 = 666;

   constructor A(int data)
   begin
```

```
        this.data = data;
    end

    method void meth1 ()
    begin
        System.out.println "meth1: abcdefghijklmnopqrstuvwxyz" + data);
    end

    method void meth2 ()
    begin
        System.out.println "meth2:" + data);
    end

    function void func ()
    begin
        System.out.println "func:" + data2);
    end

    beginMain
        var A a1 = new A(123);
        a1.meth1(); PRINTS OUT: meth1:123
        var A a2 = new A(456);
        a2.meth2(); PRINTS OUT: meth2:456
        A.func();   PRINTS OUT: func:666
    endMain
end
// END FILE:    jtw-tutorials/A.jtw
```

To move this **class** into a **package** called (for argument's sake) `pkg`, you need to set the **class**'s visibility status from none (i.e. **package** visibility) to **public**. Also each **package** visible (i.e. no **private** or **public** or **protected** specification) **class** variable, **function**, **method** and **property** needs to have its visibility status changed from **package** to **public** if you want to be able to access these items from outside of the **package**. If you have more than one **class** in the same file, they will have to be separated into separate files as you can only have one **public class** per file. Also the name of the package must be declared via a **package** specification like so **package** `pkg`; Here is the same source file, ready to be put into a **package**:

```
// BEGIN FILE: jtw-tutorials/pkg/A.jtw
package pkg;

public class A
begin
    public property int data;

    public classVar int data2 = 666;

    public constructor A(int data)
    begin
        this.data = data;
    end

    public method void meth1 ()
    begin
        System.out.println "meth1:" + data);
    end

    public method void meth2 ()
    begin
        System.out.println "meth2:" + data);
    end

    public function void func ()
```

```
    begin
        System.out.println "func:" + data2);
    end

    beginMain
        var A a1 = new A(123);
        a1.meth1(); // PRINTS OUT: meth1:123
        var A a2 = new A(456);
        a2.meth2(); // PRINTS OUT: meth2:456
        A.func(); // PRINTS OUT: func:666
    endMain
end
// END FILE:    jtw-tutorials/pkg/A.jtw
```

Also the source file for the **class** needs to be moved into the folder `~/jtw-tutorials/pkg`. To run the **class**, you will need to invoke the `Makefile` command:

```
make build pkg/A.run
```

2.16.3 Moving a class into a sub-package

Suppose you want to move a **class** A from no package (the folder `~/jtw-tutorials`) to a package called for argument's sake `pkg.inner`, the steps from the §2.16.2 needs to be followed, the only difference being that the package spec needs to be changed to **package** `pkg.inner;` and the file needs to be moved into the folder `pkg/inner`. To run the **class** file you need to invoke the following Make command:

```
make build pkg/inner/A.run.
```

Here is the **class** definition for the file `~/jtw-tutorials/pkg/inner/A.jtw`:

```
// BEGIN FILE: jtw-tutorials/pkg/inner/A.jtw
package pkg.inner;

public class A
begin
    public property int data;

    public classVar int data2 = 666;

    public constructor A(int data)
    begin
        this.data = data;
    end

    public method void meth1()
    begin
        System.out.println "meth1:" + data);
    end

    public method void meth2()
    begin
        System.out.println "meth2:" + data);
    end

    public function void func()
    begin
        System.out.println "func:" + data2);
    end

    beginMain
```

```
        var A a1 = new A(123);
        a1.meth1(); // PRINTS OUT: meth1:123
        var A a2 = new A(456);
        a2.meth2(); // PRINTS OUT: meth2:456
        A.func(); // PRINTS OUT: func:666
    endMain
end
// END FILE:    jtw-tutorials/pkg/inner/A.jtw
```

2.16.4 Importing a package

When referring to a **class** or **interface** in a package you need to specify the package name in front of every **class** name and **interface** name in the package you want to access, like so, in the main folder `~/jtw-tutorials` (outside of any package):

```
// BEGIN FILE: jtw-tutorials/B.jtw
class B
begin
    beginMain
        var pkg.A a1 = new pkg.A(123);
        a1.meth1(); // PRINTS OUT: meth1:123
        var pkg.A a2 = new pkg.A(456);
        a2.meth2(); // PRINTS OUT: meth2:456
        pkg.A.func(); // PRINTS OUT: func:666
    endMain
end
// END FILE:    jtw-tutorials/B.jtw
```

To avoid having to qualify each **class** name and **interface** name with it's package, you need to use the **import** directive like so before the definition of the **class** like so:

```
// BEGIN FILE: jtw-tutorials/B2.jtw
import pkg.*;

class B2
begin
    beginMain
        var A a1 = new A(123);
        a1.meth1(); // PRINTS OUT: meth1:123
        var A a2 = new A(456);
        a2.meth2(); // PRINTS OUT: meth2:456
        A.func();   // PRINTS OUT: func:666
    endMain
end
// END FILE:    jtw-tutorials/B2.jtw
```

2.16.5 Importing a package from another package

When referring to a **class** or **interface** in a package you need to specify the package name: **package** pkg; at the top of the file before any actual code. Where the pkg package lives in a folder called `~/jtw-tutorials/pkg`.

```
// BEGIN FILE: jtw-tutorials/pkg/C.jtw
package pkg;

public class C
begin
```

```
   beginMain
      var pkg.inner.A a1 = new pkg.inner.A(123);
      a1.meth1(); // PRINTS OUT: meth1:123
      var pkg.inner.A a2 = new pkg.inner.A(456);
      a2.meth2(); // PRINTS OUT: meth2:456
      pkg.inner.A.func(); // PRINTS OUT: func:666
   endMain
end
// END FILE:    jtw-tutorials/pkg/C.jtw
```

To avoid having to qualify each **class** name or **interface** name with it's package, you need to use the **import** directive like so after the **package** declaration but before the definition of the **class** like so:

```
// BEGIN FILE: jtw-tutorials/pkg/C2.jtw
package pkg;

import pkg.inner.*;

public class C2
begin
   beginMain
      var A a1 = new A(123);
      a1.meth1(); // PRINTS OUT: meth1:123
      var A a2 = new A(456);
      a2.meth2(); // PRINTS OUT: meth2:456
      A.func(); // PRINTS OUT: func:666
   endMain
end
// END FILE:    jtw-tutorials/pkg/C2.jtw
```

2.16.6 Modifying the `Makefile` to build a class that calls other class(es)

When your **class** X uses another **class** Y then you need to add to the `build` target which is initially like so:

```
build: clean
```

to what follows:

```
build: clean Y.java
```

If your **class** Y is in another **package** such as the **class** `~/jtw-tutorials/path/to/dir/Y.class` i.e. in the **package** `path.to.dir` then you need to add to the build target like so:

```
build: clean path/to/dir/Y.java
```

This process should be repeated for every **class** that is called, directly or indirectly from your **main** **class** X. This process can be applied to build an entire package when you simply issue the command `make build`. To actually build and run the X **class**, let `~/jtw-tutorials/path2/to/dir/X.class` be the location of the X **class**. Then you need to invoke the following `Makefile` target:

```
make build path2/to/dir/X.run
```

The "`build`" target calls the "`clean`" target which deletes all ***.java** and `*.class` files directly or indirectly in the folder `~/jtw-tutorials`. If you don't do this then **java** might run an old version of `*.class` files despite earlier errors in the build process. This is because the use of pipes in building and executing `*.class` files hides the return values of the programs **javac** and **java**.

2.16.7 Running javadoc on a package

To invoke `javadoc`, you need to issue the following command from the folder `~/jtw-tutorials`:

```
make build
```

See §2.16.6 for more information about setting up the `build` target. Then you need to issue the following command from the folder `~/jtw-tutorials`:

```
javadoc path3/to/pkg -d /path4/to/dir
```

where `path3.to.pkg` is the name of the package that you want to build and `/path4/to/dir` is the desired location for your documentation files in `*.html` format.

2.17 Passwords for the J.T.W. tutorial answers

Here are the passwords for the tutorials, which are located at the following Website:

davin.50webs.com/J.T.W

The place to enter your passwords is Section 3 of the above Web page.

No.	Password
1	policefish
2	chessweta
3	tallpencil
4	freshwhale
5	sneakermagic
6	kingpump
7	lakemarmite
8	nutriciouslamps
9	sadbutter
10	skyfresh
11	fivemagpies
12	phonesheds
13	dawnsweet
14	nightroads
15	blackscrews
16	snowfrog
17	tenflower

Chapter 3

J.T.W. Software License

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such

abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential

component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in

accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

> a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
> b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
> c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
> d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a Larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue

to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

> a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
> b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
> c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
> d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
> e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
> f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from

those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM

TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

Bibliography

[GRHV95] E. Gamma, R. Johnson R. Helm, and J. Vlissides, *Design patterns: Elements of reusable object-oriented software*, Addison Wesley, 1995.

[Sei05] Peter Seibel, *Practical common lisp*, Springer-Verlag, 2005.

Index

Praise for my book: *"Davin is bright and has a deep understanding of programming matters."*, Dr Andy Cockburn, email: andy<*at*>cosc<*dot*>canterbury<*dot*>canterbury<*dot*>ac<*dot*>nz Associate Professor of the Department of Computer Science, the University of Canterbury, Christchurch, New Zealand.

Michael Pagan, email: michael<*at*>pagan<*at*>member<*dot*>fsf<*dot*>org, said of it: *"I must say, his book is very well organised and easy to understand for a beginner like me ... Once I get deep into this book, I'd like to send him my comments. Java is such a great language and to have a book that covers it in such an eloquent way while involving Emacs in the process is too much of a rarity and a delight for me to ignore."*

This book is about how to add a preprocessor to the Java language to turbo charge its performance. Both expressiveness and efficiency can be improved using a preprocessor. The preprocessor language is called *J.T.W.* which stands for *Java Training Wheels* and is intended to make it easier for novices to program in Java. The suitability of Richard Stallman's *GNU Emacs* text editor for hosting this preprocessor language is demonstrated by examples. If you are especially clever, you can write your own Emacs Lisp **d-defmacro**s to replace blocks of tiresome repetitive "boilerplate" code in Java. A small collection of **d-defmacro**s have been written for you to deploy in your client code.

Davin Pearson was born in 1973 and is an ex-Computer Science tutor from the University of Canterbury, Christchurch, New Zealand. He has three and a half years of experience tutoring Stage I Computer Science programming courses to computer programming novices. He is probably New Zealand's foremost exponent of GNU Emacs having used it for 20 years (Happy Anniversary Emacs!) and having written over 55,000 lines of Emacs Lisp customisation code some of which he has published. While on his beloved computer he enjoys listening to music of all genres and while not on his computer he enjoys reading literature of all genres. For more information please visit his personal Website at davin.50webs.com. Photograph ©2017 Simone Pearson.

ISBN 978-0-244-92181-1
90000
9 780244 921811