Exercise 1

Java for C programmers

By the end of this exercise you will be able to

- Understand what Java has to offer over C.
- Understand the limitations of Java.

Introduction

If you are a C programmer then you are ready for a blow-by-blow account of what the Java language has to offer:

1. **Support for object-oriented programming.** Java uses the concept of a `class` which is identical to a C struct, except that in addition to containing data, classes may also contain program code. The class concept allows for support for the standard object-oriented concepts of *encapsulation*, *inheritance* and *polymorphism*. These will be covered in exercises 5, 11 and 13, respectively.
2. **Garbage collection.** The Java system keeps track of all allocated memory so that the programmer never needs to issue a `free` operation like they do in C. This means that a Java program will not be susceptible to memory leaks that can happen in a C program. It also prevents the problem where a program crashes because the programmer accidentally accessed some memory that they had already freed.
3. **No dangerous pointers!** Java's *references* are a restricted version of C's pointers, restricted to prevent the dangerous style of programming used by C programmers which although efficient, leads to programs that sometimes crash. Aspects of C's pointers that are missing from Java's references are:

   (a) The ability to add to and subtract from a pointer.

   (b) The ability to make a pointer out of any piece of memory using the `&` operator.

   (c) The ability to use casting to disable the type-checking system that is designed to keep the contents of a pointer consistent with what it is supposed to be pointing to. Java still has the concept of casting, but is no longer the back door to unsafe programming practices that it is in C. Casting is covered in exercise 12.

   Exercise 4 gives a detailed overview of how references work as the way to manipulate objects in Java.
4. **Better handling of "abnormal" circumstances.** Java introduces the high-level language concept of *exceptions* to be used in the design of programs that behave more reliably in abnormal circumstances. The C approach is for functions to return a "funny" value when something goes wrong, like how the `fgetc` function returns `-1` in the event that you are attempting to read past the end of the file. One of the problems with this approach is that the user of the `fgetc` function might forget of check the return value, and then an unexpected end of file would cause the program to behave unpredictably. Exercise 10 covers exceptions.

5. **More standardisation.** The C language allows for a lot of things to vary from one C compiler to another, and from one system to another.

   (a) The size of the numerical data types is not fixed, whereas in Java they are fixed so that the result of a numerical computation will be the same, no matter what system it is run on.

   (b) The order of evaluation of some expressions is not fixed in C, so that the result of evaluating `++x + x` cannot be known for sure without compiling it on a particular system. With Java this ambiguity does not exist.

6. **Guaranteed initialisation.** All data is guaranteed to be initialised to a known value before it is first used, In C, the use of `malloc` and uninitialised local variables allows random initial values to sneak into a program, depending on what happens to be in the computer's memory at the time.

7. **Bounds checking on arrays.** Java checks every array access at run time that it is within the bounds of the array. Accessing a C array out of bounds can cause the program to crash or behave unpredictably. In Java, the worst that can happen is that an `ArrayIndexOutOfBoundsException` will be thrown. Arrays are covered in exercise 9.

8. **No unsafe unions.** The use of the union concept in C leads to programs that can crash if the programmer fails to keep track of what type of data the union currently contains. Java's type checking system has a run-time component as well as the conventional compile-time component that C has. The use of run-time type enquiry in Java allows you to do exactly what unions did in C, but without the risk of the program crashing. The worst that could happen in Java with run-time type enquiry is that an exception will be thrown. Run-time type enquiry is covered in exercise 12.

9. **Built-in documentation.** Comments that start with `/**` are extracted by a program called `javadoc` for use in documenting the behaviour of the class. See exercise 5 for an example of this.

10. **No need to maintain two files for each library of code:** the `.h` header file and the `.c` source file. With Java, everything is contained within one `.java` file.

11. **No need to keep separate the two concepts of *declaration* and *definition*.** In C, variables and functions often need to be both declared (usually in the header file) and defined (in the source file). In Java, there is no distinction between the two concepts, so we can speak simply of the *definition* of something.

12. **Definitions can be in any order.** In C, functions and variables needed to be declared earlier in the source file than where they were used. The Java compiler is smarter and for example, allows you to use a variable that is defined somewhere later on in the source file.

13. **Local variables can be defined at the point of first use.** Local variables in C must be defined at the top of a function or a brace-enclosed block. Related to this is how the syntax of the `for` loop has been changed to allow the counter variables to be defined in the first part of the `for` loop like so:

    ```
    for (int i=0; i<10; i++) {
       System.out.println(i);
    }
    ```

    which is equivalent to the following C code:

```
{
   int i;
   .
   .
   .
   for (i=0; i<10; i++) {
      System.out.println("%d\n",i);
   }
}
```

14. **No global variables or global functions.** Everything in Java must be defined within a class. Functions in java are called *methods* and are called using a syntax that emphasises the role of the object:

    `object.method(` *`arguments...`* `)`

    To get the equivalent of globals in Java, make `static` methods and `static` data within a class.

15. **No goto statement.** Java introduces a better-behaved system of *labelled breaks* for when gotos would have been required.

16. **No need to link object files into one large executable.** Java's `javac` compiler compiles `.java` source files to files ending in `.class` and the linking is done at run time, with classes linked into the program as they are needed.

17. **A boolean data type.** Programmers of C are forced to use integers to store the values true and false. Having a boolean data type allows the compiler to catch more errors at compile time.

18. **Better support for strings.** Having strings as instances of the `String` class allow them to be garbage collected like any other object so the programmer doesn't need to worry about freeing them.

    The `String` class contains a whole collection of useful functions for manipulating strings, including a length method so that `s.length()` returns the length of the string `s`. For adding strings, there is a built-in operator `+` so that `"Hello"+" there!"` returns the concatenated string `"Hello there!"`.

19. **Two styles of comments.** Java borrows the `//` style of comments from C++ and some C compilers.

20. **No preprocessor.** The features of the C preprocessor such as constants, inline functions and `#include` directives are all supported directly within Java, so there is no need to use a preprocessor for these. The feature of conditional compilation is not supported by Java so conditional compilations would have to be expressed by `if` statements in the language itself.

21. **Method overloading** allows you to re-use a method name for different argument types. For example the method `System.out.println(...)` can accept arguments of any type. The appropriate `println` method will be chosen by the compiler at compile time depending on the number and types of the arguments. See exercise 3 for more detail on this.

22. **Better security.** There is a special kind of Java program called an *applet* that has strong limitations on what it can do, such as accessing the files on your computer. They are suitable for use over the Internet, where you don't necessarily trust the person who wrote the applet not to attempt to corrupt your system or defraud you by stealing personal information like your credit card number.

23. **Built-in support for multi-threading** which allows you to create applications that are more responsive to user input.

The main disadvantage of Java is its lack of speed compared to C. This is the price of the extra standardisation and more stable behaviour. As computers get faster, speed should be less of an issue and the gap between C and Java is closing as the state of Java technology improves.

Another disadvantage of Java is the way it is not possible use it to access the hardware directly. To overcome this limitation, Java has the `native` keyword which provides a way of executing code that is written in other languages such as C or assembly where this limitation does not apply.

Questions

1. Look at the Java program listing in exercise 2 and attempt to write a C version. You may not be able to complete this question until you have completed exercise 2 for an understanding of how methods, properties and constructors work.

   (a) Create a header file called `Person.h` which should contain a definition of a `Person` struct containing two members `name` and `age`. Use the following typedefs so that your C code will look similar to the Java code:

   ```
   /* useful Java-like abbreviation for "char *" */
   typedef char *String;

   /* useful abbreviation for "struct Person *" */
   typedef struct Person *PersonPtr;
   ```

   The header file should also contain the following function declarations:

   ```
   void talk(PersonPtr p);
   void commentAboutAge(PersonPtr p);
   PersonPtr make_person(String aName, int anAge);
   ```

   The `make_person` function is supposed to do the job of the Java program's `Person` constructor.

   (b) Create a file called `Person.c` containing the definitions of these three functions and also a `main` function that tests them out in a similar way to how the `main` method of `PersonTest` tests out the `Person` class.