Exercise 9

Arrays

By the end of this exercise you will be able to

- Define an array.
- Access the elements of an array.
- Pass an array as an argument to another method.
- Write a method that accepts an array as an argument.
- Understand the difference between *object arrays* and *non-object arrays*.

Introduction

Arrays are a useful tool that appear in most programming languages. The goal of this exercise is to teach you how they are done in Java. To use an array in Java, you must first define it like so:

```
int[] fred = new int[10];
```

This defines a new array of ten integers that can be accessed using the reference `fred`. Initially the values of the ten elements of the array will all be zero, so it is often necessary to initialise the array before you use it:

```
for (int i=0; i<10; i++) {
   fred[i] = i;
}
```

This sets the elements of the array to the numbers one through ten. If the size of the array to be created is known at compile time like with the `fred` array, then the following can be used in place of the previous four lines of code:

```
int[] fred = {1,2,3,4,5,6,7,8,9,10};
```

This single line defines the `fred` array and initialises the elements of the array and should be a familiar technique to C programmers. For non C programmers, this code is probably less clear than the above so I will only mention it here, and not use it in the rest of the examples. You can now access the elements like so:

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

This prints the contents of the array to the screen. In the last two `for` loops we accessed the elements of the array using the square bracket notation that will be familiar to programmers of C and Pascal, but it is possible to leave out the brackets and just say `fred` by itself, as in the following example:

```
ArrayPrint.printArray(fred);
```

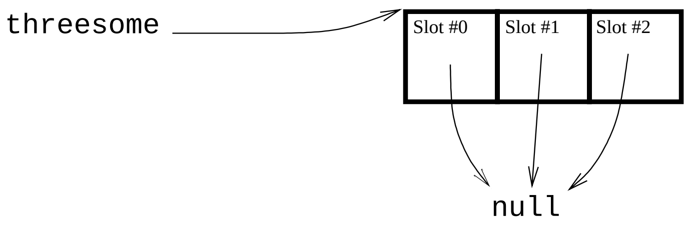
This passes a reference to the `fred` array to the `printArray` method of the `ArrayPrint` class which causes the contents of the `fred` array to be printed to the screen like in the previous example.

The `fred` array is an example of a *non-object array* because it is an array of one of Java's non-object types: `int`, `long`, `short`, `byte`, `float`, `double`, `char` and `boolean`. The other kind of array is the *object array* which is an array of object references. This will be covered in question 2.

Questions

1. **Non-object arrays.**
   (a) Fetch the file `ArrayTest.java` and load it into your editor.
   (b) Add a line to the `doStuff` method to create a new array called `nums` of ten floating-point numbers. Use the `double` type for double-precision floating-point numbers.
   HINT: Look at how the `fred` array was created and copy the pattern for this.
   (c) Use a `for` loop to set the elements of the `nums` array to the values 1.0, 1.1, 1.2, through to 1.9.
   (d) Write a second `for` loop to print out the elements of the `nums` array to make sure that the values are what they are supposed to be. Compile and run the program to find out.
   NOTE: Java's `double` and `float` types are incapable of storing some numbers exactly, so there may be some inexactness in the values that are printed out for this question.
   (e) Fetch the files `ArrayPrint.html` and `ArrayPrint.class`, then load the `html` file into your web browser. This web page contains the javadocs of the `ArrayPrint` class which contains two static `printArray` methods. Place a call to one of these methods to print the contents of your `nums` array. Hopefully this will produce similar results to the previous question.
   (f) Now write your own `printArray` method in the `ArrayTest` class that does the same job as the corresponding method in the `ArrayPrint` class.
   HINT: Your `printArray` method should be general-purpose enough to work with `double` arrays of any given length. To do this you will need to access the `length` property of the array object. If your array is called `x`, then the length of this array is given by: `x.length`.
   (g) Add a call to the `printArray` method that you have just written and verify that it produces the correct output for the `nums` array.

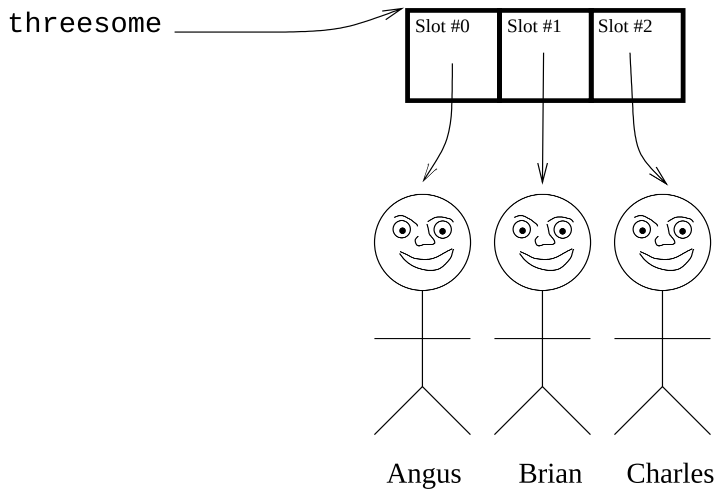
2. **Object arrays.**
   (a) Fetch the file `ArrayTest2.java` and load it into your editor.
   (b) In the `doStuff` method, define an array of three `Human` objects called `threesome`.
   HINT: This is almost identical to what you did to create the `double` array in question 1b, except that the class `Human` appears in place of the type `double`.
   An object array like `threesome` is actually an array of object references, so that when you create an object array, the elements of the array are all references to the value `null`, meaning "no object". Here is a diagram of this relationship:



(c) Create three new `Human` objects with names Angus, Brian and Charles and assign them to the `threesome` array.

HINT: Unlike question 1c, you can't use a loop for this.

Here is how the array looks after this code has been written:



(d) Add a `for` loop to print out the three `Human` objects using each object's `toString` method.

HINT: This is very similar to question 1d.

(e) Write a `printArray` method that works like the `printArray` method that you wrote in question 1f, except that this time it accepts `Human` arrays as an argument.

HINT: Copy the code from question 1f, but replace every occurrence of `double` with `Human[]`.

(f) Call the `printArray` method that you wrote in the previous question, passing the array `threesome` as the argument.