Exercise 6

Static versus non-static

By the end of this exercise you will be able to

- Understand the difference that is made by the `static` modifier.
- Call a static method.
- Access a static property.

Introduction

What difference does the `static` modifier make? From the previous exercises you will be familiar with its use in the `main` method but when else can it be used?

The purpose of this exercise is to show you what the static concept means in Java. In this exercise, there is one example which shows you what happens if you label all your methods and properties as static and another example which shows you what happens when just one property is static and the rest non-static.

Questions

1. Fetch the file `CartoonTest.java`, which contains some classes designed to represent the cartoon characters from the TV series *The Flintstones*.

2. Add some code to the `main` method to call Fred Flintstone's `displayMe` method.

   HINT: Remember that to call a normal (non-static) method, you put the name of the object before the name of the method. To call a static method, you put put name of the class in front of the method.

3. Add some code to the `main` method to print out Barney Rubble's favourite colour *without* using Barney Rubble's `displayMe` method.

   HINT: To access a static property, put the name of the class in front of the property.

4. In the three Flintstone character classes shown above every method and property was static, mainly to show you an example of using the `static` keyword. When everything is static you don't need to create any objects, instead you can "talk" to the classes directly.

   We will now re-work the previous example by merging the three Flintstone classes into one `CartoonCharacter` class:

```
class CartoonCharacter {

   // Properties of the class...
   private String name;
   private String favouriteColour;
   private int    favouriteNumber;
```

```
    // Constructor of the class...
    public CartoonCharacter(String aName, String aColour, int aNumber) {
        name            = aName;
        favouriteColour = aColour;
        favouriteNumber = aNumber;
    }

    // Methods of the class...
    public void displayMe() {
        System.out.println("Hello, my name is " + name);
        System.out.println("my favourite colour is " + favouriteColour);
        System.out.println("and my favourite number is " + favouriteNumber);
    }
}
```

The code for this class is contained within the file `CartoonCharacter.java` which you shoul fetch now. Delete all of the code in the `main` method of `CartoonTest.java` and add three lines of code to construct three `CartoonCharacter` objects called Fred Flintstone, Wilma Flintstone and Barney Rubble.

5. Call `displayMe` method of class `CartoonCharacter` for each of the objects you created in the previous question.

6. Add a line of code to the `main` method to print out the favourite colour of the Barney Rubble object that you created in question 4.

   HINT: Since the `favouriteColour` property of the `CartoonCharacter` class is in a different class from the `main` method, you will need to change it from `private` to `public`.

7. In the questions so far in this exercise, you have seen an example where static was used but the example was quite artificial because we then saw how we could re-work it so that nothing was static. This question will show an example where static is genuinely useful.

   We would like to keep a record of how many `CartoonCharacter` objects we have created. To achieve this, add a `public static int` property called `count` to the `CartoonCharacter` class.

   In the `CartoonCharacter` constructor, add a line that increments the value of `count`.

   Finally, at the end of the `main` method, add a line to print out the value of `count`, which should be `3` when you run the program.

   Why must `count` be a static property for it to properly count the number of cartoon characters created? Try removing the `static` keyword and seeing what happens.

8. Another use for static is for constants. Since the value of a constant is never changed, you might as well have one value for the entire class rather than a different value for each object.

   An example of this is the `PI` property of the `Math` class which stores the well-known number 3.14159... from mathematics. Add some code to the `main` method to print out the `PI` property to the screen.