Programming 2:
Tutorial 3

Set by: Luke Dickens

14th – 18th Oct, 2019

Reminder about the tutorial sheets

Remember that the best way to learn a programming language and understand the concepts is to do lots of programming. This involves a good deal of problem solving, and that requires you to think, experiment and test things. Please look at all the questions spend some time thinking carefully about them, before asking for help. If you are still stuck:

- ask the module leader, or the lab helpers, for help at the lab sessions
- or post a question on the moodle course page

Some questions in the early labs (typically the first 3) are marked with a [*] symbol. **These questions are compulsory** and **you will be assessed** on one or more of these in the following week's lab.

Questions marked as [!] or [!!] are there to make you think. You only need to *sketch* a solution to these, and model solutions may not be provided. Do not worry if you cannot complete these harder questions without help.

Getting Started

For these tutorial questions there are some supporting files that can be found on moodle. Download the zip file on moodle and place in a folder on your machine: these notes will refer to this folder as `<root>`. Each question will tell you which *subfolder* your files are in. For example, if you unzip the file in folder `<root>`, then any files for a question in subfolder `example` can be found at location: `<root>\tutorial3\example\`.

You must also learn how to navigate the command line and compile and run your programmes. Please look at Lecture 1 materials on the Taught Content tab of the Moodle page for some videos to help you get started with this.

1 Any and All

Comment: *This question is not assessed, but may help you get started on the assessed questions.*

Look at the `AnyAll` class in the `any_all` folder and complete the following steps.

a) Complete the class method declared in this class with the following signature:

```
public static boolean any(boolean[] values)
```

This method should take a single array of booleans as argument and return `true` if **any** of the elements of this array are `true`, and `false` otherwise.

Comment: *You can do this with a standard for loop, but I would like you to do this using the for-each syntax. Look at the lecture slides to remind yourself if needed.*

b) How can you tell this is a class method by looking at its declaration?

c) Test your code with the `AnyAllTester` programme in the same folder.

d) Now write a second `static` method for the `AnyAll` class called `all`, which will be similar to the `any` method.

The method `all` should take a `boolean` array as an argument, and return a `boolean`. It should return `true` if **all** elements of the input are `true`, and `false` otherwise.

e) Add some test code to `AnyAllTester` that will check that `all` behaves as you expect it.

2 The Collatz Conjecture [*]

a) Look at the programme `Echo` in the directory `collatz`. Can you predict what the programme does? Compile it and run it with:

```
java Echo
```

Can you explain the output? What will happen if you run it instead with:

```
java Echo fish
```

or with:

```
java Echo fish 58
```

For this last run of the programme:

i. What value does `args[0]` take? What is its type?

ii. What about `args[1]`?

b) "...1 4 2 1 4 2 1..." In the same folder there is an incomplete programme `Collatz`, open it up and take a look. Add a `public` **class** method called `next`, which takes a single integer, n, as input and returns an `int`. It should return:

- $\frac{n}{2}$ if n is even
- $3n + 1$ if n is odd

Hint: *you may wish to use the modulo operator,* `%`*. This takes two operands and finds the remainder of the first divided by the second. For instance,* `11 % 5` *will evaluate to* `1`*. Note also that* $3n$ *is mathematical notation for* 3 *times* n*. How do you write this in java?*

c) Do you need to create an object of type `Collatz` in order to call the `next` method? Can you explain why?

d) Now complete the `main` method, so that the `Collatz` program:

- First, takes a positive integer command-line argument as an initial value and displays this value
- Then, stores the value in a variable and repeatedly applies `next` to the variable, displaying each value that is computed

The program should terminate when the value 1 is computed. For instance, given the input 37 your program should print:

```
37 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

Remember that the main method of your program has an argument `args` that is an array of type `String`. Thus the first command-line argument can be accessed via `args[0]`. To turn this argument into an integer, you can use the `parseInt` method of the `Integer` class. E.g.:

```
int integerVersionOfFirstArgument = Integer.parseInt(args[0]);
```

Comment: *Fun fact: It is unknown whether this program will always terminate (for* any *initial integer). However, the* Collatz conjecture *posits that it does.*

Submission: You should submit `Collatz.java`.

3 Counting Flips [*]

Look at the `Coin` class in folder `counting_flips`. This is the same class you used in previous tutorials.

a) Add an instance method `totalNumberOfFlipsForThisCoin()` to the class `Coin`. This should return the total number of times the method `flip()` has been called so far for each coin *separately*. Then test this with the program `CoinProgWithFlipCountInstance` provided in the same file. You should get output along the following lines:

```
PROGRAM STARTED
COINS CREATED

STATUS: first coin 1 flips; second coin 1 flips:
First coin: heads, second coin: tails.

Type:
  f - to flip only the first coin
  s - to flip only the second coin
  b - to flip both coins at once
  q - to quit

Enter selection: f

STATUS: first coin 2 flips; second coin 1 flips:
First coin: heads, second coin: tails.

Type:
  f - to flip only the first coin
  s - to flip only the second coin
  b - to flip both coins at once
  q - to quit

Enter selection: b

STATUS: first coin 3 flips; second coin 2 flips:
First coin: heads, second coin: heads.

Type:
  f - to flip only the first coin
  s - to flip only the second coin
  b - to flip both coins at once
  q - to quit

Enter selection: q

STATUS: first coin 3 flips; second coin 2 flips:
First coin: heads, second coin: heads.
PROGRAM ENDED
```

Hint: *To do this you will also have to add an* instance *variable to act as a counter.*

Why are there 3 flips counted after just a single "f" entered by the user?

b) Now add a class method `totalNumberOfFlipsForAllCoins()` to the same `Coin` class. This method should return the total number of times the method `flip()` has been called so far *for all the coins*. Then test this with the program `CoinProgWithFlipCountStatic` in the same folder. You should get output along the following lines:

```
PROGRAM STARTED
COINS CREATED

STATUS: first coin 1 flips; second coin 1 flips (2 total).
First coin: tails, second coin: heads.

Type:
  f - to flip only the first coin
  s - to flip only the second coin
  b - to flip both coins at once
  q - to quit

Enter selection: s

STATUS: first coin 1 flips; second coin 2 flips (3 total).
First coin: tails, second coin: heads.

Type:
  f - to flip only the first coin
  s - to flip only the second coin
  b - to flip both coins at once
  q - to quit

Enter selection: b

STATUS: first coin 2 flips; second coin 3 flips (5 total).
First coin: tails, second coin: heads.

Type:
  f - to flip only the first coin
  s - to flip only the second coin
  b - to flip both coins at once
  q - to quit

Enter selection: q

STATUS: first coin 2 flips; second coin 3 flips (5 total).
First coin: tails, second coin: heads.
PROGRAM ENDED
```

Hint: *To do this you will also have to add a class attribute to act as a counter. You define a class attribute using the* `static` *keyword.*

What is the difference between the values returned by the methods totalNumberOfFlipsForThisCoin() and totalNumberOfFlipsForAllCoins()?

Can you explain why the first method is an instance method and the second is a class method?

c) Now write/draw a UML description of your `Coin` class. Make sure that you indicate which fields and methods are instance methods, and which are `static`.

Submission: You should submit `Coin.java`, `CoinProgWithFlipCountInstance.java` and `CoinProgWithFlipCountStatic.java`.

4 Balloon Factory [*]

Look at the `Balloon` and `BalloonFactory` classes in folder `balloon_factory`. `Balloon` objects can be `"Red"`, `"Green"` or `"Blue"` but the constructor does not take an input argument. Instead, the

colour of the balloons is determined by class attributes. Complete the following steps, but **do not** add any other constructors or edit the existing constructor.:

a) The defined attributes `alternates`, `allColours` and `index` should be class attributes. The attribute `colour` should be an instance attribute. Modify the attribute definitions to ensure this. What types are these attributes?

b) Look at the constructor, and the `toString` and `incrementIndex` methods. Can you describe what each does? Look at Blocks A & B in `BalloonFactory`: what will happen if this code is compiled and run as it is written? Check your prediction.

c) Add a `public` class method `switchAlternates` to `Balloon`. This takes no inputs and returns nothing. The method should modify the `alternates` class-attribute: from `false` to `true` or from `true` to `false`.

d) Uncomment Block B of the `main` method in `BalloonFactory`. How will this change the output? Check your prediction.

e) Add a `public` class method `createBalloons` to `Balloon`. This takes a single `int` input, `number`, and returns an array of `Balloon` objects. The method should create an array of size `number` and fill each element with a new `Balloon` object.

f) Uncomment Block C of the `main` method in `BalloonFactory`. Use `createBalloons` to initialise the `balloons` variable. Compile and run your programme.

g) How would you modify Block C to create a second array of only `"Green"` balloons.

Submission: You should submit `Balloon.java` and `BalloonFactory.java`.