Programming 2:
Tutorial 2

Set by: Luke Dickens

7th – 11th 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>\tutorial2\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 Decimal Numbers [*]

Look at the `DecimalNumber` class in folder `decimal`.

a) Look at the class `DecimalNumber`. How many constructors does it have? What is the difference between them?

b) Add a new constructor to the class `DecimalNumber` that takes a `double` input setting the attribute `value` to the value of the input.

c) Add a new constructor to the class `DecimalNumber` that takes another `DecimalNumber` as a single argument, and returns a copy of the original `DecimalNumber`; this is sometimes called a *copy constructor*.

d) Look at the `DecimalNumberTester` class, can you predict what the output will be? Compile and run the code to see if you're predictions are correct.

What constructors are being used on line 10? What about line 11? And line 12?

e) What line is the setter method defined on? How is this different from the constructor you defined in part b).

f) Uncomment lines 18-22, then compile and run the `DecimalNumberTester` again. You should get the following output:

```
The value of num1 starts as 10.0.
The value of num2 starts as 20.0.
The value of num3 starts as 20.0.

The value of num1 ends up as 30.0.
The value of num2 ends up as 30.0.
The value of num3 ends up as 20.0.
```

g) Add a comment at the end of `DecimalNumberTester.java` to explain why it produces this output.

h) Look at the commented code on line 24. This reads:

```
//num2.value = 45;
```

What will happen if the code is uncommented? Why?

Submission: You should submit `DecimalNumber.java` and `DecimalNumberTester.java`. Note that `DecimalNumber.java` should contain your edits from Questions 1 and 2, so do not submit it until you have also completed Question 2.

2 Comparing Decimal Numbers [*]

You are again working with the code in folder `decimal`. You will be editing the classes `DecimalNumber` and `ComparingDecimalNumbers`.

a) Look at the code in `ComparingDecimalNumbers.java`. What is the earliest line containing

i. an `int` literal

ii. a `float` literal

iii. a `double` literal

iv. a `String` literal

b) Of the literals above, which are passed as arguments for methods, and which as used as arguments in operations?

c) Compile the `ComparingDecimalNumbers` class. Does it compile? If not, what do the errors mean?

d) Add a `public` method `lessThan` to `DecimalNumber`. This should take another `DecimalNumber` as input and return a `boolean`. If the `value` of `this` object is less than the `value` of the input `DecimalNumber`, it should return `true`. Otherwise, it should return `false`.

e) Compile and run `ComparingDecimalNumbers`. Can you explain the output.

f) Uncomment block B, then compile the `ComparingDecimalNumbers` class. Does it compile? If not, what do the errors mean? How what could you add to `DecimalNumber` so that `ComparingDecimalNumbers` compiles?

g) Add an overloaded `public` method `lessThan` to `DecimalNumber`, which takes a `double` as input and again returns a `boolean`. If the `value` of `this` object is less than the input `double`, it should return `true`. Otherwise, it should return `false`.

h) Uncomment block C, then compile the `ComparingDecimalNumbers` class. Can you explain the error? Are there any changes you can make to DecimalNumber that will allow block C to compile? Write your explanation in a comment at the end of the file.

Submission: You should submit `DecimalNumber.java` and `ComparingDecimalNumbers.java`. Note that `DecimalNumber.java` should contain your edits from Questions 1 and 2, so do not submit it until you have also completed Question 1.

3 Party Drinks [*]

Look at the `Drink` and `PartyDrinks` classes in folder `party_drinks`.

The `PartyDrinks` class contains a main method and a helper method representing the events at a DIS party. Try compiling `Drink`. Does it compile? Try compiling `PartyDrinks`. Does it compile? If either class fails to compile, what do the errors mean?

You will complete the `Drink` class to make the `PartyDrinks` programme run properly. Implement the class to meet the following requirements:

a) Add a `String` attribute called `flavour` and an `double` attribute called `level` to `Drink`. Ideally these should be declared `private`

b) Add a constructor which takes a single `String` argument called `flavour`. The constructor body should assign the input argument `flavour`, to the attribute `flavour`. The `level` attribute should be set to 0. Ideally, this should be declared `public` (as should your other methods).

c) Add a method called `fill`, which takes no input arguments and returns nothing (declaration). This method should simply set the `level` attribute to 1 (implementation).

d) Add a method called `consume`, which takes a `double` as input called `amount` and returns nothing. When called:

- if `level` is less than or equal to the input `amount`, it should set `level` to 0.
- if `amount` is less than than `level` but greater than 0, then `level` should be set to `level - amount`.
- in all other cases `level` should remain unchanged.

Hint: *There are two challenges associated with implementing the constructor. The first is to declare it properly (see the* `Car` *class on the slides for some examples). The second is to implement it (write the body). Remember, to distinguish an attribute from an input argument of the same name, you can use the* `this` *operator.*

Hint: *When writing an instance method from scratch there are again two challenges. First you must write the declaration, then the implementation. Start by looking at the slide on* ***Anatomy of a Method****. Remember, you must declare the return type of your method or use* `void` *to indicate it does not return anything.*

Submission: You should submit `Drink.java` and `PartyDrinks.java`.

4 Scallop Cards [!]

This is a longer question than those presented previously and may take you a little longer to work through. It is designed to give you experience of working at a slightly larger scale, as well as looking at how classes can work together. It will not be assessed.

Look at the `Journey`, `ScallopCard` and `ScallopCardProb` classes in folder `<root>\tutorial2\scallop_cards`.

`Journey` is a simple class designed to represent a journey on public transport. The `ScallopCard` class represents a basic payment card to use on public transport. `ScallopCardProg` is a simple programme that uses the other two classes `Journey` and `ScallopCard`, to capture the effects of adding some balance to the payment card then using the card for journeys.

You will notice that some of the code in `ScallopCard` and `ScallopCardProg` has been commented out. This is so that you can compile and test your answers to earlier questions. You will have to *uncomment* some code as you get to the appropriate questions.

a) Add a constructor and four getter methods to the `Journey` class. The definitions should match the usage of these getter methods in the `toString` class provided.

b) Should you add setter methods to the `Journey` class? Explain your answer.

c) Uncomment the code in `ScallopCard` and `ScallopCardProg` marked for question part c). Next, add getter methods to the `ScallopCard` class as well. Try to compile and run the code. Does it work as expected?

d) Begin by uncommenting the code in `ScallopCardProg` marked for question part d).

   Methods which change the state, but do not necessarily directly set it, are sometimes called *mutator methods*. Write a mutator method named `addBalance` for the class `ScallopCard`, that changes the `balance` field. This should take an `int` argument called `supplement`, and add this to the current balance. The method should not return anything.

Hint: this method needs to be used by other classes, so it should be **visible** *to them. Also, this method changes field values of an* **instance** *of the* `ScallopCard` *class, so should it be a* `static` *method or not? Finally, it doesn't return anything, so what should it's return type be?*

Try to compile and run the code. Does it work as expected? Why might `addBalance` be preferred to a direct setter method?

e) We will not be adding a mutator method for the `cardId` field. Why not?

f) Uncomment the code in `ScallopCard` and `ScallopCardProg` marked for question part f). Try to compile and run the code. Does it work as expected? In the class `ScallopCard`, there is a mutator method that changes the `journeys` field of called `addJourney`. Look at this method, can you predict what it will do?

Add a method to the `ScallopCard` class called `canTravel` that takes a `Journey` argument, and returns `true` if there are enough funds on the card to cover the journey, and `false` otherwise. Should you make this an instance or a `static` method? Should this be `public` or `private`? Make sure your method definition reflects these choices.

g) In the `main` function, add four more work-days of travel identical to the one already added, using the `addJourneyGroupToCard` method.

Now, add a night out to the card too. This should consist of the following journeys:

- From Ladywell to New Cross by Bus costing $1.10
- From New Cross to Shoreditch by Overground costing $1.10
- From Shoreditch to London Bridge by Bus costing $1.30
- From London Bridge to Lewisham by Bus costing $1.30

You should create a `Journey` array containing the four `Journey`'s, then use the `addJourneyGroupToCard` method.

h) The cost of a `Journey`, and the balance on a `ScallopCard` are quite similar. Both are defined as `int` variables representing the total value in pence. When included as part of a `String`, they are converted into pounds and pence (see the `toString` methods in `Journey` and `ScallopCard`). What might be a better way of dealing with these similaritites?