Programming 2: Tutorial 6

Set by: Luke Dickens

12th – 16th Nov, 2018

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 try all the standard questions (those not [hard] or [harder]), and spend some time thinking carefully about them, before asking for help. If you are still stuck:

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

Questions marked as *harder* 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 the harder questions without help.

1 Factorials and Triangular Numbers

A number of mathmatical concepts can be defined in a recursive fashion. Here we take advantage of those definitions. As usual, the provided code for this question is in the appropriate folder of the provided code archive.

a) Look at the `FactorialCalculator` class. The `fact` method calculates the factorial of the input argument, and is the same code as in the lectures. However, most mathematicians would also consider `0` to have a factorial of `1`, i.e. `0! == 1`. Run the code and see what happens if you try to calculate the factorial of `0`. Can you explain this behaviour?

b) Now try to correct the code so that `0!` is also calculated correctly. You should also write preconditions and postconditions in comments.

   Hint: you only need to change one line of the code. Is it the base or the standard case you need to change?

c) The triangular number is a very similar concept to a factorial. The nth triangular number, T_n, is the sum of all positive numbers equal to or less than n, i.e.

$$T_n = n + (n-1) + \ldots + 1 \qquad (1)$$

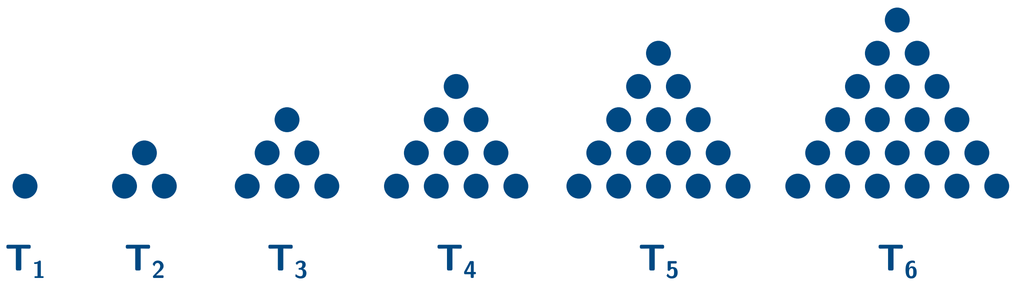

Figure 1: A graphical representation of the first six triangular numbers.

To see why these numbers are called triangular numbers see Figure 1.

Now write another class called `TriangularNumberCalculator` using the `FactorialCalculator` class as a template. Your new class should have a `main` method, and one recursive method called `triangularNumber`. This should be `static`, and take a single `int` argument `n`. The method should return an `int` that is the triangular number of the input `n`.

The `main` method should be very similar to the `FactorialCalculator` class and should request an integer input from the user then output the triangular number of that input.

Hint: Before thinking about coding, begin by deciding what the base case is, then decide what the standard case is in conceptual terms. Only then should you try to translate that to the code.

2 Sums and Products

In previous tutorials, you have written methods to calculate the sum of elements in an array using a loop. Here you will try to do the same thing with recursion.

a) Look at the class `SumsAndProducts`, and consider, the two overloaded methods called `sum`. The first method takes a single argument of type `double` array, and returns a sum of the elements in the array. To do this, it calls the second method. **Do not edit the first `sum` method!**

The second `sum` method takes two arguments, a `double` array, and an `int` called `from` – which refers to a position in the array. This second method should sum only those elements in the array with position greater than or equal to `from`. For instance, if you pass in the array `data` equals {`1.1, 2.2, 3.3`} and the position `from` equals `1`, it should return `5.5`. Your job is to complete this method definition. You should write this definition as a recursive method.

Test your answer with the programme `SumsAndProductsProg`.

Hint: You should again think about your base case and standard case. Remember, in your recursive call, some input variables will change but some can remain the same.

b) Now write a similar pair of methods to take the product of elements in an array of `double`s. (Put these methods in the same class.)

A product over an array should multiply the first element by the second element, then multiply the result by the third element, and so on to the end of the array. For instance, if you take the product of the array `data=={2.0,3.0,5.0}` you should get the result `30.0`.

Hint: you should have two overloaded methods, the first should take one argument, and the second two arguments. Your method with two arguments should be recursive.

c) Edit the code in `SumsAndProductsProg` so that the user first inputs the array, and then decides whether they want its sum or product. The programme should print the output of the requested operation.

d) [harder] Look at the `SimpleVector` class from tutorial 3. Can you write a method `sum` that takes an array of `SimpleVector`s and sums them all?

3 Sentence and Word Reversing [optional]

Recursion can be a very elegant way of reversing sequences of elements, even when you don't know how many objects you have in your sequence.

a) Look at the class `WordCounter`. This is an example of a recursive method given in the lectures, that counts the number of words typed in by the user. Using this class as a template complete a new class `StringReverser` in the same package, which will take a user's input sentence and print it to screen backwards.

   Your job is to write a `reverseSentence` method. The `reverseSentence` method should be `public` and `static`, take a single input argument of type `Scanner`, and should have `void` return type. Instead of returning anything, this method will simply print the words read from the `Scanner` object, in reverse order, to the screen.

   Your `main` method simply has to request a sentence as input then call `reverseSentence` passing the `Scanner` as input argument. If you are unsure about this, look again at the `main` method in the `WordCounter` class.

   Hint: Think carefully about the order of things in your standard case here. The base case should check that the end of the sentence has been reached. You may find the `endsWith` method from `String` helpful here. The `substring` and `charAt` methods from `String` may also be helpful. Look these methods up in the documentation if you are not sure how to use them.

b) [harder] Now write a new class, `StringReverser`, with two methods, a `main` method and a recursive method called `reverseString`. `reverseString` should take a string as input, and output a `String` which is the input in reverse. When run, the programme should request a word from the user, and then print it out in reverse. For example, one run of the programme could look like this:

```
> java tutorial6.question3.StringReverser
Type in a word: pupils
pupils in reverse is slipup
```