9.7 Summary and Exercises

In this chapter we looked at *methods.* A *method* is a way of grouping together code. This makes it possible to re-use it but that is only part of the role of a method: they are also vital to helping us write structured, understandable and (usually) shorter code.

A method has:

- A **name** which identifies it.
- A **return type** which identifies the type of data returned by the method (this can be void if no data is returned).
- A possibly-empty **list of parameters** and their types, identifying the data passed to the method for computation.
- An optional list of **modifiers** (of which **static** is one).

Parameters are passed to methods as *copies* - so it is not possible to change the value of simple data items using methods directly. Because complex data is managed using *references*, and only the *references* are copied when passed as parameters, although it isn't possible to change the references themselves, it *is* possible to change the data they refer to.

Self-Test Quiz

Complete the progress test to see what you know - this is not assessed and I can't see the result (only if you have completed it). You can attempt it multiple times *but it would be better if you attempt it first WITHOUT referring to the notes to see how much you remember*.

Chapter 9 Progress Test Quiz (https://canvas.swansea.ac.uk/courses/44525/assignments/293483)

Exercises

The first set of exercises on methods are based on going back to exercises on previous sheets and turning them into methods. As well as being ‘right’ methods need to have sensibly-chosen parameters (number and type), and return values. Remember methods should do one thing - not several.

From Exercise Sheet up to Chapter 3

1. **Turn your program that converts mm to inches into a method mmToInches** - make sure it has the correct number and type of parameter(s), and returns the right type.
2. Remember the program that printed this:

```
                    +-+-+
                    | | |
                    +-+-+
                    | | |
                    +-+-+
```

Turn it into a method printGrid that accepts an integer parameter saying how may ‘boxes’ high and wide the shape should be.

In the example above the value of the parameter is two). Extend your method so it takes two parameters - for length and width -

so it can print rectangular (i.e. not just square) grids.

3. **Write a method called isEaster that accepts three parameters - all integers - representing a year, a month, and a day.** Your method should return true if the year, month and day correspond with Easter (the algorithm is on Exercise Sheet 1).
4. **Challenge** - do 3 but instead of integers representing year, month and day, the parameter should be a Java Date object.

From Exercise Sheet up to Chapter 4

5. **Write a method numDigits that accepts an integer as a parameter and returns the number of digits it has** (for example, 99 has 2, 1664 has 4).

6. **Challenge** - redo 5 above but with an additional parameter that represents the base to use for the number. So, for example, 99 in base 2 is 1100011, which is 7 ‘digits’ long; 15 in base 16 is F, which is 1 ‘digit’. Remember that only certain integers are legal/sensible values for a base and do checks before any calculations.

7. **Redo exercise 7 on the exercise sheet for chapter 4 as a method called toGrade which converts a letter grade to a UK university grade** - think about the types you should use.

8. **Similarly, redo exercise 8 on the same sheet as a method toCardName** - again think about the types and error handling.

9. **Challenge** - actually should be quite easy by now! Redo exercise 16 on the chapter 4 sheet as a method isBlack that returns true if a chessboard square is black and false otherwise. Think about what parameters you should use.

10. **Write a method gcd that computes the greatest common divisor of two numbers (integers) and returns it.**

11. **Challenge** - actually not so much of one by now. Write a method called isLegalCardNumber that returns true if a credit card number is ‘legal’ (that is potentially valid). The algorithm to check this is in exercise 8 on the sheet for Chapter 6..

12. **Write a getInteger method that reads in an integer from the keyboard without crashing** - your method should keep asking for an integer until the user types one - ignoring all non-integer input. It should print appropriate messages to the user.

13. **Extend your method from 12 above to only accept positive integers** (>= 0 in this case).

14. **Extend your method from 13 above to allow the user to cancel by typing c** (think carefully about what your method should return in this case.

15. **Extend your method so that the strings printed for the user are passed as parameters**; add a test program that demonstrates your method working in at least two different languages. If you are not bilingual, Google Translate is your friend - I'm not going to be fussy about minor translation errors.

Exercise Sheet up to Chapter 6

16. **Write a method that converts an integer to a string representing the day of the week.** You can choose if Sunday is day 1 or 7, but your method should do something sensible if you pass a parameter outside the range of 1 to 7.

17. **Extend your program so that it accepts a second parameter which is a language name, and returns the name of the day of the week in the corresponding language** - your method should work for at least two languages, one of which can be (but doesn't have to be) English. If you don't know any other languages, Google translate is again your friend but to start you off:

Days of the Week

English	**Welsh**	**Greek**
Sunday	Dydd Sul	Κυριακή
Monday	Dydd Llun	Δευτέρα
Tuesday	Dydd Mawrth	Τρίτη
Wednesday	Dydd Mercher	Τετάρτη
Thursday	Dydd Iau	Πέμπτη
Friday	Dydd Gwener	Παρασκευή
Saturday	Dydd Sadwrn	Σάββατο

Exercise just about Methods

Write *two* methods (in *addition* to any main method you write) that deal with Roman numerals.

If you don't use two methods AS WELL AS the 'main' method this task will not be considered correct. Also you must comment your methods properly, as described in Stage 5.

The first method should take *either* a string consisting of one character, *or* (better, easier) a *character* (char) and convert it to its numeric value. Either:

public static int romanNumeralToInt(String romanNumeral)

or (better)

public static int romanNumeralToInt(char romanNumeral)

The values of Roman numerals are:

I = 1

V = 5

X = 10

L = 50

C = 100

D = 500

M = 1000

(all other letters should return zero)

Once you've done this, write *another method* that *uses your first* one to convert a sequence of Roman numerals represented as a string into their decimal value. For example

VI = 6; CLX1 = 161, MMXIII = 2013

Hints: Use a *switch statement* in your first method.

There is no need to worry about things like IV representing 4 - just use IIII; similarly don't use IX to represent 9 - instead just use VIIII Using IX, IV etc. is called *subtractive notation* - the Romans didn’t consistently use subtractive notation themselves so I don’t see why we should:-).

You might find these methods useful:

- The string method **substring(x,y)** which returns the part of the string starting at character x and ending at character y-1. So if a string value is equal to “banana split” then **value.substring(0,6)** is “banana”.
- The string method **length()** which tells you how long a string is
- The string method **charAt(x)** which returns the character (char) at location x