Laboratory 1

In this lab you'll be working individually, but if helps you learn then feel free to discuss your approach with others.

The instructions will start out quite detailed, but get progressively sparser -- if you're struggling or something is unclear then please ask a TA and they should be able to give you a pointer.

Typos/errors? Email sarah.clinch@manchester.ac.uk and I'll get them fixed.

Exercise 1

In the lecture, we introduced the following code to demonstrate functionality of Java's switch statement.

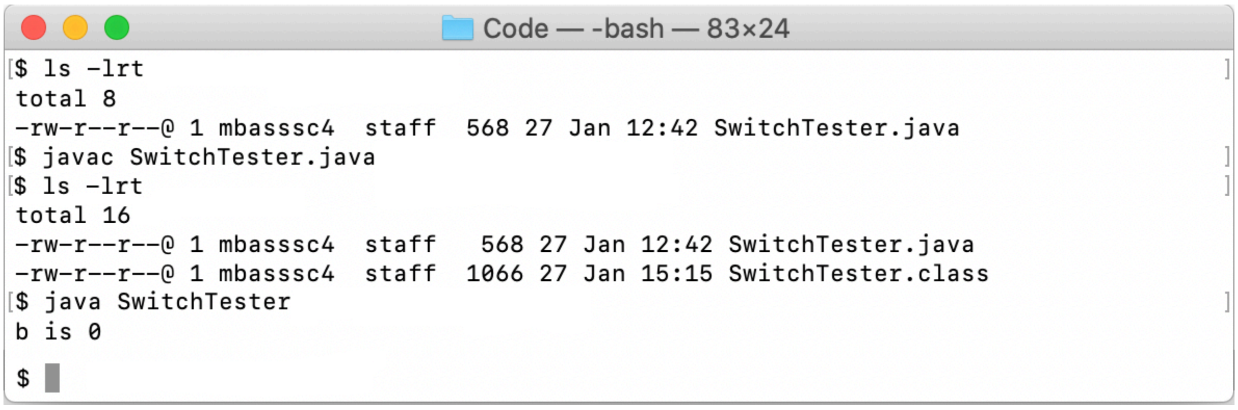
```
class SwitchTester {
    public static void main(String args[]) {
        int a = 8;
        int b = 8;
        switchExperiment(a, b);
    }

    public static void switchExperiment(int a, int b) {
        switch (a) {
            case 1:
                b -= 4;
            case 2:
                b += 4;
            case 3:
                b /= 4;
                break;
            case 4:
                b *= 4;
                break;
            default:
                b = 0;
                break;
        }
        System.out.println("b is " + b);
    }
}
```

Step 1

Save the above code snippet to a file called `SwitchTester.java`. Compile the code using the `javac` command (`javac SwitchTester.java`).

List the contents of the directory in which you compiled the Java file -- can you identify where Java has written the compiled bytecode?

The compiled bytecode should be in a file called `SwitchTester.class`. You can run this file using the `java` command (`java SwitchTester`).



```
$ ls -lrt
total 8
-rw-r--r--@ 1 mbasssc4  staff  568 27 Jan 12:42 SwitchTester.java
$ javac SwitchTester.java
$ ls -lrt
total 16
-rw-r--r--@ 1 mbasssc4  staff   568 27 Jan 12:42 SwitchTester.java
-rw-r--r--@ 1 mbasssc4  staff  1066 27 Jan 15:15 SwitchTester.class
$ java SwitchTester
b is 0
$
```

Note that Java requires the class name in the file and the filename itself to match. Remove the `SwitchTester.class` and then rename your file to `switchTester.java`.

Compile the newly-renamed Java file: `javac switchTester.java`. What happens? Has it generated a new class file? If so, how is it named? Does the file run?

Step 2

Let's start by experimenting with Java's types.

First, what type is it anyway? Java doesn't provide a clean way of accessing the type of a primitive value, but the following line of code should tell us what we want to know:

`System.out.println(((Object)a).getClass().getName());`

Add this code after line 5, which should be the call to `switchExperiment(a, b);`. Make another copy of the line directly below the one you have just introduced, and swap the variable `a` for `b`. Your new main method should look like this:

```
    public static void main(String args[]) {
        int a = 8;
        int b = 8;
        switchExperiment(a, b);
        System.out.println(((Object)a).getClass().getName());
        System.out.println(((Object)b).getClass().getName());
    }
```

Compile and run your modified code. You should get the following output:

```
b is 0
java.lang.Integer
java.lang.Integer
```

Although this isn't the primitive type `int` that we were hoping for, it's pretty clear that both a and b are integers. That's all we needed to know.

Modify the code to change the type of `a` to be `short` and `b` to be `byte`. Before compiling and running the code, what do you think will happen? Do you expect to need to change any other parts of lines `3` and `4` for this code to run successfully?

Run the code, what just happened?

So `int`, `byte` and `short` are all used to hold whole numbers. For this reason, the statements in lines `3` and `4` are equally valid with any of the three types. Try modifying line `4` to change the value of `b` to 128. Does it still compile and run OK? What would be the maximum accepted values of the three different types?

If you've made any edits as you experimented with the three whole number types, make sure you restore your main method back to this:

```
    public static void main(String args[]) {
        short a = 8;
        byte b = 8;
        switchExperiment(a, b);
        System.out.println(((Object)a).getClass().getName());
        System.out.println(((Object)b).getClass().getName());
    }
```

So we now understand why lines `3` and `4` are OK, but what about line `10`? Running the above code works fine (assuming you've copied it correctly), and produces the following output:

```
b is 0
java.lang.Short
java.lang.Byte
```

If Java is strictly typed, why is it allowing `a` and `b` to be something other than the `int` type? Why does the code run successfully?

If two data types are compatible then, in execution of statements such as these, Java will impicitly convert one type to another. This is refered to as ***automatic type conversion****. For the variables to be automatically converted, two things must be true: firstly, the two types must be compatible with each other; and secondly, the type being converted must be smaller than the resulting data type (e.g., int to long). The following conversions will all work:*

```
byte -> short -> int -> long -> float -> double
```

By contrast, if the type being converted is larger than the resulting data type, then automatic type conversion cannot occur as it would result in a loss of precision.

Step 3

During the lecture, we established a truth table for the `switch` statement. Let's revisit (and extend) that now. For each row in the table, record in the rightmost column the value of `b` when the `a` value on line `3` is set equal to the leftmost column. Do not adjust the value of `b` on line `4`.

a	b	switchExperiment(a, b);
1	8	2
2	8	
3	8	
4	8	
5	8	

Without adjusting `a` or `b`, correct the code so that it produces the following results:

a	b	switchExperiment(a, b);
1	8	4
2	8	12
3	8	2
4	8	32
5	8	0

Modify the code so that the call to `switchExperiment(a, b);` (line `5`) is contained inside a **single** `for` loop such that you can populate the following truth table:

a	b	switchExperiment(a, b);
1	1	-3
2	4	8
3	9	2
4	16	64
5	25	0

Step 4

So far you have been working with the *colon and break* version of the switch statement. Since preventing all fall-through with breaks is the common case, the use of `break` becomes verbose and prone to making mistakes, as you probably learned. The *lambda arrow* notation replaces the colon and the break making the code more readable and with further features. The following two blocks are equivalent:

Colon and break

```
switch (a) {
    case 1:
        b -= 4;
        break;
    case 2:
        b += 4;
        break;
(...)
}
```

Lambda arrow

```
switch (a) {
    case 1-> b -= 4;
    case 2-> b += 4;
(...)
}
```

Unlike the *colon and break* notation, multiple statements withing each case must be within braces `{` `}`. Update your solution from Step 3 using the Lambda arrow notation.

Step 5

Write your own `switch` statement that operates over a `String` variable. The program should take a month name as input, and print the season that the given month is contained within. In addition, the month January should print "New Year, New Me!", and August "Summer Vacation!".

Test your code by calling the method with a variety of different month values.

Optional extension - To test your code without lots of edits and recompiles, write a loop to iterate over an array of months, calling the method for each entry in the array.

Step 6

It's often helpful to understand the limits of a new command. Here are some things to try, which does Java allow you to do with a `switch` ? Which ones do you predict will work? What behaviour do you expect? Were your intuitions correct?

1. A `switch` that works over a boolean value
2. A `switch` that works over a float value
3. A `switch` whose cases are of different types
4. A `case` that contains multiple statements
5. A `switch` where integer cases appear in an arbitrary order
6. A `switch` statement without a default
7. A `case` value that is determined by a constant (keyword: `final`)
8. A `case` value that is determined by a variable
9. A `case` value that is determined by a value (ie an arithmetic operation `2*5`)
10. A `case` with multiple values
11. A `default` case with a `break`
12. A `switch` statement where one or more of the `case` labels are duplicated
13. A combination of `case`s with `->` and `:` operators
14. A `case` with no content after the colon
15. A `switch` expression with a `null` expression (i.e. `null ->`)
16. A `switch` expression with NO `null` expression that is passed a `null` value.

Exercise 2

In order to do this exercise you will have to:

- Understand Java methods (Chapter 5 of the book)
- Define a 2D array (Section 6.9 of the book)

If you haven't read the above book sections you may struggle to complete this exercise. It is encouraged to have the book open during the lab.

A magic word square is a square in which a word can be formed by reading each word and column. For example:

```
L I M B
A R E A
C O R K
K N E E
```

(All word square examples in this exercise are taken from Wikipedia).

Step 1

- Create a new class: we don't know what classes are yet but you can get some inspiration from the `SwitchTester` class in Exercise 1.
- Declare and initialise a 2D array to hold the word square above.

Step 2

Write a `static` method `printRow` that takes two parameters: a word square, and the index of the row to be printed. When called, the method should print the word within the specified row.

Test your method by asking it to print the 1st row of the square, is the result as expected?

Step 3

Write a `static` method `printColumn` that takes two parameters: a word square, and the index of the column to be printed. When called, the method should print the word within the specified column.

Test your method by asking it to print the 4th column of the square, is the result as expected?

Step 4

Declare and initialise two more 2D arrays for the following word squares:

```
T O O     S C E N T
U R N     C A N O E
B E E     A R S O N
          R O U S E
          F L E E T
```

Test your two methods `printRow` and `printColumn` with the new arrays and a variety of values? Do they still work as expected -- if not, amend your code until it will work with squares of different sizes.

Step 5

A diagonal word square is one in which the two main diagonals are also words.

```
B A R N
A R E A
L I A R
L A D Y
```

Write a `static` method `printDiagonal` that takes two parameters: a word square and a boolean that indicates whether to print the top-left-to-bottom-right diagonal or the top-right-to-bottom-left diagonal.

Step 6

Some word squares use the same words over the rows and columns. For example:

```
A    N O    B I T    C A R D    H E A R T    L A T E R A L S
     O N    I C E    A R E A    E M B E R    A X O N E M A L
            T E N    R E A R    A B U S E    T O E P L A T E
                     D A R T    R E S I N    E N P L A N E D
                                T R E N D    R E L A N D E D
                                             A M A N D I N E
                                             L A T E E N E R
                                             S L E D D E R S
```

First test your code with each of the above squares, does it still work for squares with these new sizes?

Step 7

Write a new `static` method `wordsRepeated` to test if a square is using the same words over the rows and columns.

You should be sure to use a different kind of loop to those you have used so far (e.g. if you've only been using `for` loops in previous steps, you should use a `while` or `do... while` for this exercise.

To see if two strings are equal, you will need the [`.equals()`](#) *method. e.g.* `"string 1".equals("string 2")`*. You'll learn more about this method in Week 3.*

Step 8

Word rectangles are similar to word squares, but the horizontal and vertical words are of a different length.

```
F R A C T U R E
O U T L I N E D
B L O O M I N G
S E P T E T T E
```

Do your `printRow` and `printColumn` methods handle these out-of-the-box or are some changes needed?

Can you update your `printDiagonal` and `wordsRepeated` so that they don't print output if they are called on a word rectangle, but they continue to work as expected when called with a word square?

All done!

Solutions will be uploaded to Blackboard on Monday Week 2, around 5pm. +++++++++++++