COM1027 Programming Fundamentals

Lab 5

Repetition Statements and the use of Strings

Purpose

The purpose of this lab is to develop your skills and ability to:

- Make use of the repetition statements (loops),
- Use nested loops with arrays
- Understand how Strings are created
- Understand the difference between identity and quality of Strings
- Introduce Regular Expressions

Lab Structure

Labs are a mixture of step-by-step instructions that enable you to learn new skills, and exercises so that you can define your own examples.

We will use **Concept Reminders** in front of text when we are reminding you of concepts you have already learnt previously in labs or lectures.

In using this document it is possible to paste text directly into Eclipse. That means when defining pieces of code contained in lab sheets in Eclipse their values can be copied directly from here, which will remove the possibility of mistyping values. If you are reading a PDF version of the document, then please click on the 'Select' icon that activates text selection. Before running the code, ensure that the code has been copied correctly, and make the relevant changes to make the code fully functional. Sometimes, some hidden or incorrect characters could be copied to Eclipse.

The lab exercise at the end of this document will be **assessed**. They are marked, and contribute to your final grade. This lab exercise has 10 marks to earn which will contribute marks towards your final grade. Every place you have to add code is indicated as an 'Exercise' with instructions.

A total of 10 labs will be assessed, which will correspond to the first 20% of your final grade (2% of each lab). Please submit your completed work using GitLab by Week 12.

You must comply with the University's academic misconduct procedures:
https://www.surrey.ac.uk/office-student-complaints-appeals-and-egulation/academic-misconduct

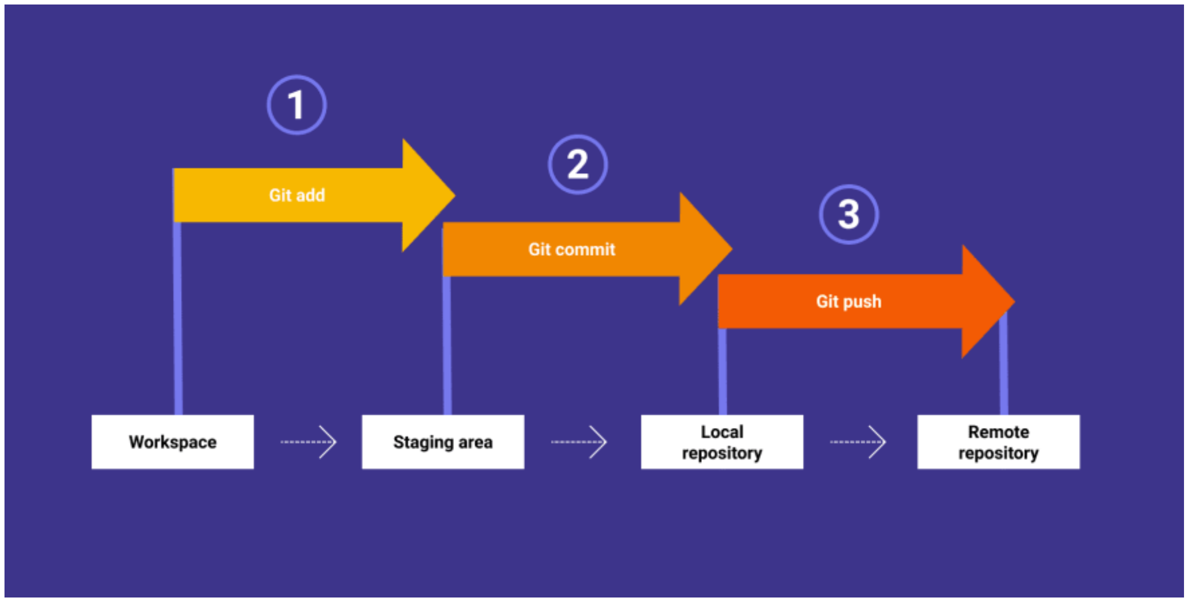
Please ask us lots of questions during the labs, or use the discussion forums on SurreyLearn. All the demonstrators are here to help you get a better understanding of the Java programming concepts.

Instructions

For this module, we will be using automated assessment tools for all lab activities and coursework. To achieve this, we will be using the Faculty of Engineering and Physical Sciences (FEPS) GitLab platform (https://gitlab.surrey.ac.uk) to upload your completed code.

In last week's lab activity, you completed the following:

- Downloaded a project from SurreyLearn and imported it into your workspace
- Made changes to the project by creating new classes and objects
- Used the UML diagrams to convert the structured English language to Java code
- Commited the changes to your local repository and
- Pushed those changes onto your remote repository on GitLab



Screenshot taken from `https://docs.gitlab.com/`

In this lab, you will download a new project from SurreyLearn, make substantial changes to the project, commit and push those changes to your remote repository.

Important Note: Do not push any buggy code to the repository. Use the `'Run As > Java Application'` or `'Run As > JUnit Test'` as indicated in the instructions below. You should only 'Commit and Push' code when it works.

Reminder: You can access your GitLab repository via `https://gitlab.surrey.ac.uk/`, using your IT username and password.

Example

In this example, we will use the debugging tool to inspect instance variables and see in detail how the values change in each line of code.

1. Import an existing Eclipse project and set your workspace

From SurreyLearn, download the project called `COM1027 Lab05.zip`. Extract the contents of the project in the Downloads folder. Copy the uncompressed (unzipped) `Lab05` folder **to your local repository**. **Note:** Do not copy the `COM1027_Lab05` folder. Navigate in the folder, and only copy `Lab05` folder.

If you are using the computer labs, then your local git directory would look like this:
`/user/HS***/[username]/git/com1027[username]/`
For example:
`/user/HS223/sk0041/git/com1027sk0041/`
where `sk0041` shows a sample username.

Start Eclipse and open the same lab workspace as last time. Import the project in Eclipse, by clicking on `File > Import > General > Project from Folder or Archive`. Select `Directory` to locate `Lab05` from your local git repository. Once the files are loaded, select the Eclipse project and click `Finish`.

Once imported, run the `Example1` class from the `lab5` package as a Java application (right-click on `Example1.java` and select `Run As - Java Application`. A set of values will be displayed on the console.

Look at the class `Example1` which has two arrays: `names`, `grades`.

```
String[] names = new String[3];
double[] grades = new double[3];
```

The application displays all the elements of both arrays on the console. Please try to change the code so that you make use of repetition statements (**for** loops).

Concept Reminder:
A `for` statement specifies that a statement of block of statements is to be obeyed several times.

```
for ([initialisation]; [condition]; [increment]) {
     .....
}
```

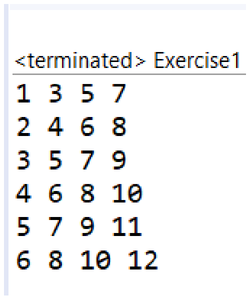
2. Developing an understanding of nested loops

Look at the `NestedLoops` file from the `lab5` package and run the application. You will see that the inner loop is run 12 times for each index i.

```java
// Loop through the first 5 times tables.
for (int i = 1; i <= 5; i++) {

    int total = 0;

    // Print the ith times table for multipliers 1 to 12.
    for (int j = 1; j <= 12; j++) {
    // We can make use of both index i and j inside this code block.

        System.out.println(i + " * " + j + " = " + (i * j));
        // Add the value to the running total.
        total = total + (i * j);
    }
    // Print out the total value of the multiples of this table.
    System.out.println("Total of table is " + total);
}
```

The code contains two loops which then print out the first 5 times tables (with multipliers from 1 to 12). A running total of the values of each table is also kept. This is achieved by nesting one loop inside another.

Inspect the code, and use the comments to guide you through the several iterations. Now you should write a `new nested loop` in the `main` method in the `Example2` class. Based on your understanding of `for` loops and nested loops, you will now need to create the following:



```
<terminated> Exercise1
1 3 5 7
2 4 6 8
3 5 7 9
4 6 8 10
5 7 9 11
6 8 10 12
```

Hints:

- Each line follows the same pattern but from a different starting point.
- You can print to the console without start a new line using the command `System.out.print("...");`
- You can print a new line to the console just using `System.out.println();`
- You will need to refer to the outer index inside the inner loop.

If you are still struggling to get this output, consider using the following outer loop first. You can then decide how to display the second, third and forth column and what type of calculations are required in the inner loop to display the numbers 3, 4 ... 11, 12.

`for (int i = 1; i <= 6; i++)` **{** // inner loop and additional code to be added here **}**

3. Experimenting with other types of loops

An array contains a number of variables. The number of variables may be zero, in which case the array is said to be empty. The variables contained in an array have no names; instead they are referenced by a unique index. These variables are also called elements, and must have the same data type. Arrays are of fixed length. You can not change the size of the arrays once they are created. Memory is allocated to an array during its creation only, much before the actual elements are added to it. One of the benefits of arrays is that they are capable of storing objects, and this gives the capability of storing a class's object and its attributes in the array's elements.

Similarly, array lists can also contain a number of variables, and the number of variables may increase or decrease during the runtime of the application. Each element in the array list can be referenced by its unique index, just like arrays. However, array lists are more dynamic than arrays, as they are re-sizeable. This means that the size of the array lists is not fixed and they can grow and shrink dynamically.

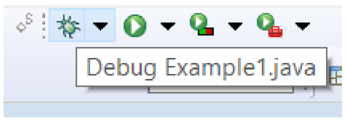
The next example makes use of array lists. Look at `Example3`. An array list has been declared that holds only String objects.

```java
List<String> names = new ArrayList<String>(20);
```

The initialisation of the array list is done through the use of a `Scanner` object that is responsible for reading in names from the command line. To achieve this, a `while` loop has been used to continue reading the input entered on the command line. The `while` loop will only end when the boolean value of `finished` is set to `true`.

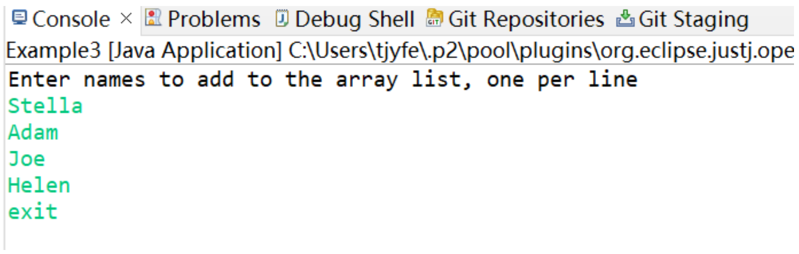
```java
 Scanner scanner = new Scanner(System.in);
...
 while (!finished) {
           String line = scanner.nextLine();

           // Check for exit.
           if (line.startsWith("exit"))
             { finished = true;
           }
           else {
             // Not exit, so add the name to the array list.
             names.add(line); // Toggle breakpoint here.
             finished = !scanner.hasNextLine();
           }
```

Prior to completing the functionality of the program, we are going to run the program but using the debugger. Set a break point on the line of code (line 42) which adds a new element with value (`line`) to `names`. To do this, right-click in the margin of the code and choose `Toggle Breakpoint`. Run the code again in Debug mode by clicking on the icon, and select `Example3` like the picture below.

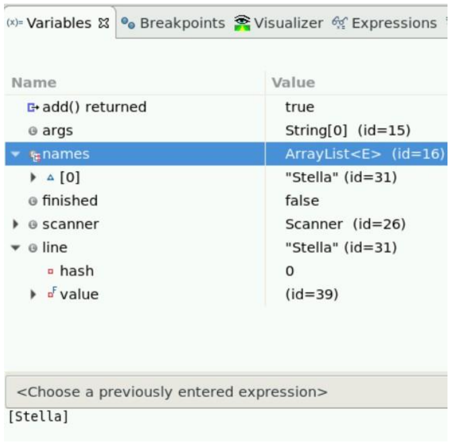


If you cannot locate the Debug icon, go to the 'Run' tab on the top of your screen, and then select 'Debug' from the drop-down menu.

Confirm that you wish to open the debugger perspective, and open the `Variable` perspective so that you can see the changes to the array list. After starting the debugger, enter a few names on the command line, so that we can move to line `names.add(line)` in the program and initialise the array list



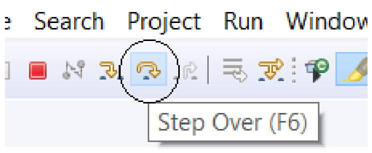
```
Example3 [Java Application] C:\Users\tjyfe\.p2\pool\plugins\org.eclipse.justj.ope
Enter names to add to the array list, one per line
Stella
Adam
Joe
Helen
exit
```



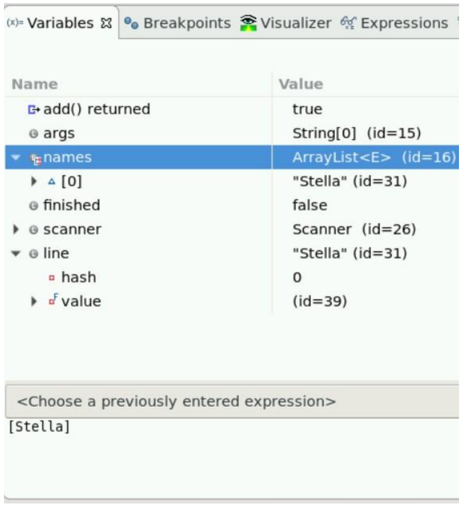
Name	Value
add() returned	true
args	String[0] (id=15)
names	ArrayList<E> (id=16)
[0]	"Stella" (id=31)
finished	false
scanner	Scanner (id=26)
line	"Stella" (id=31)
hash	0
value	(id=39)

```
<Choose a previously entered expression>
[Stella]
```

We are now at the point where we add the value of line to the first available index in the array list. You will see that the line is highlighted in green. Step over the code using the icon:



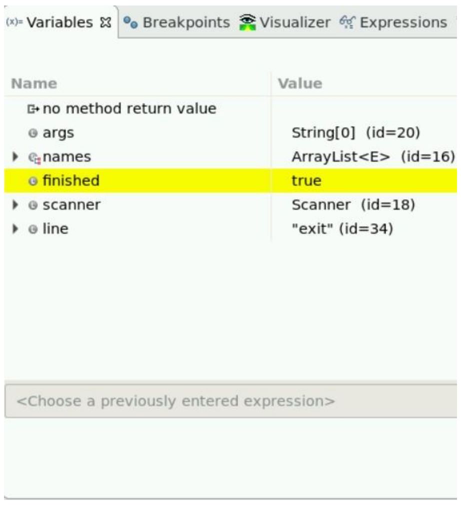
By doing this, the array list gets initialised with the value of `line` at index 0, and this can also be seen in the debugging window. Also the green highlight has moved down a line.



Name	Value
add() returned	true
args	String[0] (id=15)
names	ArrayList<E> (id=16)
[0]	"Stella" (id=31)
finished	false
scanner	Scanner (id=26)
line	"Stella" (id=31)
hash	0
value	(id=39)

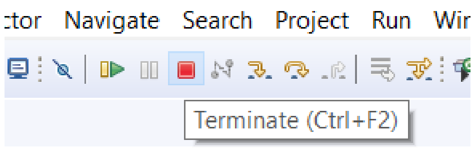
```
<Choose a previously entered expression>
[Stella]
```

Continue to step over each line of code. When prompted to enter a new value, enter `exit`, and continue to step over.



Name	Value
no method return value	
args	String[0] (id=20)
names	ArrayList<E> (id=16)
finished	true
scanner	Scanner (id=18)
line	"exit" (id=34)

This will cause the program to exit the `while` loop and continue with the rest of the code. Now terminate the running of the code using the red square icon:



Click on the Java perspective in the top right to return to the package and code view.

It is now time to complete the functionality of the class, by adding a block of code to loop through the array list and print out all of its values. To do this, add either a `while` or a `for each` loop, and structure your output like this:

```
Name is Stella
Name is Adam
Name is Joe
Name is Helen
```

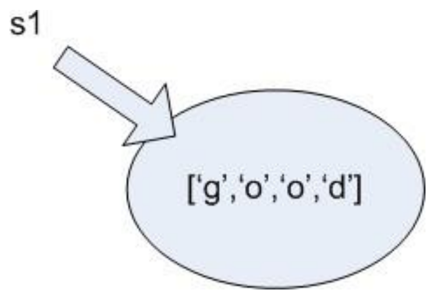
To test your work, run the Java project as a Java Application, and initialise the array list by entering values on the command line. Then type `exit` and check whether the values are correctly printed on the console.

4. Developing an understanding of Strings

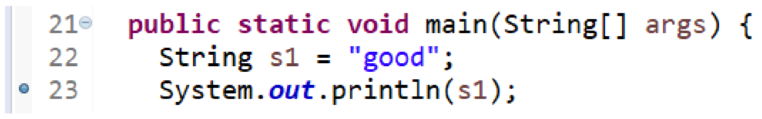
For this example we will be using `Example4`. Look at the following code:

```
String s1 = "good";
System.out.println(s1);
```

Explanation: This very simple program creates a string called `s1` using a string literal and prints it out. Using `string literals` is one of the most common ways to create strings. A string literal consists of zero or more characters enclosed in double quotes. In more detail a string literal references an object of type String. So in memory what you really have is:

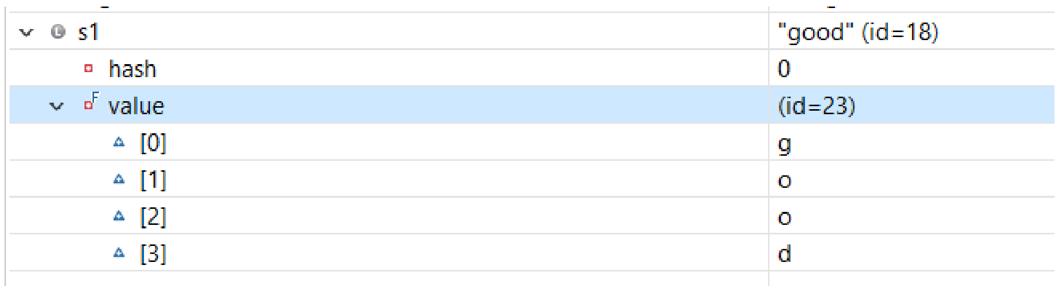


Add a breakpoint as follows and run the code again using the debugger:



```
21  public static void main(String[] args) {
22    String s1 = "good";
23    System.out.println(s1);
```

When you inspect the variable `s1` in the debugger you will see:

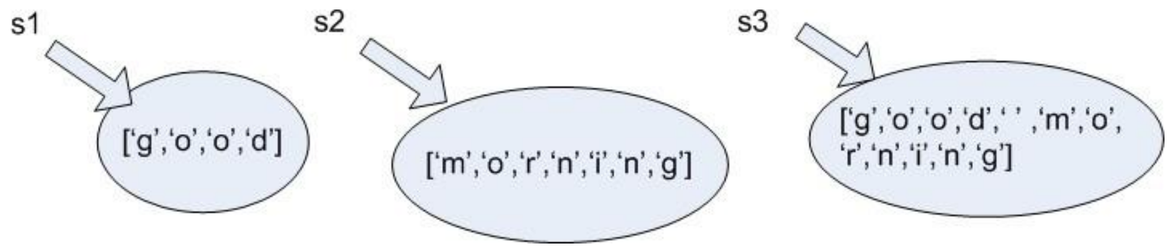


s1	"good" (id=18)
hash	0
value	(id=23)
[0]	g
[1]	o
[2]	o
[3]	d

Look at the next part of the code:

```
String s2 = "morning";
String s3 = s1 + " " + s2;
System.out.println(s3);
```

This time we have created a string `s3` which is a concatenation of other strings: `s1`, a string with a space and `s2`.



Explanation: We now have three different String objects `s1`, `s2` and `s3`. However, in creating these three strings, we have actually created lots more because the line

```
String s3 = s1 + " " + s2;
```

will create temporary strings when it is concatenating `s1 + " " + s2` together.

We can test the length of a string by using one of the methods provided by the `String` class. This is done on lines 28 − 34. An example is:

```
System.out.println(s1.length());
```

We can get the character at a particular position in a string. Add the following code and run the program again:

```
System.out.println(s1.charAt(0));
System.out.println(s1.charAt(1));
System.out.println(s1.charAt(2));
System.out.println(s1.charAt(3));
```

Both these methods (`length` and `charAt`) are part of the `String` class. Note that the indexing of the sequence of characters starts at 0. What would happen if we tried to find the value of the character at a position greater than 3?
Add the following code and run the program again:

```
System.out.println(s1.charAt(4));
```

You get an exception because the index 4 being passed means that you have tried to access a value beyond the end of the String. Remove this line of code so that we can carry on with the lab sheet.

We can now check the difference between identity and quality of Strings. Part of the code has already been written for you:

```
boolean start = false;
// Creating a string literal
String exercise = "COM1027";

// Creating a string using parameterised String constructor.
String exercise2 = new String("COM1027");
```

Test whether the `exercise` and `exercise2` variables begin with the letters, C, O and M. Add one (or two) print statements which does this checking. Look up the `String` class to help you find suitable method(s) to use. Direct link to API:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

(Note: Scroll down and check the list of methods available in the Method Summary section)

Do they both start with “COM”, despite being defined differently? (String literal vs String object) Does this make them the same?

Continue adding code in the `Example4` class. Use two conditional statements to check the identity and equity of those two String values.

It is common to compare the value of two strings. Add a conditional statement that checks if the two Sting values are equal, using the `.equals()` built-in method. Part of the code has been provided for you. Modify the code below and add it in the `main` method

```
// Does exercise equal exercise2?
if (){
     System.out.println("The two strings are equal");
}
else {
     System.out.println("The two strings are NOT equal");
}
```

Explanation: `.equals(Object anObject)` is a method in `String` class. The method is invoked on one string object, `exercise`, and the other string object, `exercise2`, is passed as a parameter to the method. (It is similar to the `toString()` method because it is an overridden method of `Object`.) The `.equals` is typically used within an if statement because the return value of the method is a `boolean` value.

Now modify the following code and add it after the first conditional statement:

```
if () {
     System.out.println("exercise equals exercise2");
}
else {
     System.out.println("exercise does NOT equal exercise2");
}
```

Both `exercise` and `exercise2` contains the same content, so why are they not equal when we use ==?

Explanation: This is because `exercise` and `exercise2` are different objects and == is testing whether their object references (their address in memory) are the same, not their content. We need to use `.equals` to compare their content. This is a common mistake, and you need to be careful when comparing String values. Now try the following:

```
String exercise3 = "COM1027";

if (exercise == exercise3)
     { System.out.println("exercise equals exercise3");
}
else {
     System.out.println("exercise does NOT equal exercise3");
}
```

Here we are still testing whether `exercise` and `exercise3` point at the same object and the answer is that they are indeed the same (immutable) object. Why is this? Java is clever enough to work out that there is only need to store one copy of the string literal “COM1027” in memory because it is defined at compile time. Therefore there is only one copy of any unique string literal in memory, and both `exercise` and `exercise3` point to that location in memory.

5. Simple regular expressions

Regular expressions are patterns that can be applied to strings to search for matches. The simplest way of doing this is to use the `.matches` method from the String class.

Regular expressions are very powerful and can be used to match any pattern of characters. Look at the following to see more:

- `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/regex/Pattern.html`
- `http://docs.oracle.com/javase/tutorial/essential/regex/char_classes.html`

Some useful resources for understanding how regular expressions are structured:

- `https://github.com/zeeshanu/learn-regex`
- `https://www.tutorialspoint.com/java/java_regular_expressions.htm`

Define a class called `Example5` which includes a main method. Add the following lines of code to the main method:

```java
// Define a reference number which must start with "HO" followed by 4 digits.
String reference = "HO1234";

boolean foundMatch = reference.matches("(HO)([0-9]{4})");
if (foundMatch) {
    System.out.println("Reference number is valid");
}
else {
    System.out.println("Reference number is NOT valid");
}
```

This code defines a `String` object containing "HO1234". This `String` is then validated against a regular expression. The required pattern is 6 characters long. The string must start with "HO" and is then followed by any 4 digits.

The regular expression to check for this pattern is `(HO)([0-9]4).` This is broken up into two groups:

1. `(HO)`: specifies that the first two characters must be exactly ``` ``HO" ```.
2. `([0-9]4)`: specifies that the next 4 characters can be any of the digits 0 through 9 `[0-9]`.

You do not have to put the "()" but it is sometimes useful to see the different parts of the pattern you are looking for by using "()".

Change `reference` to "MO1234" and run the program again as a Java Application. This time no match will be found.

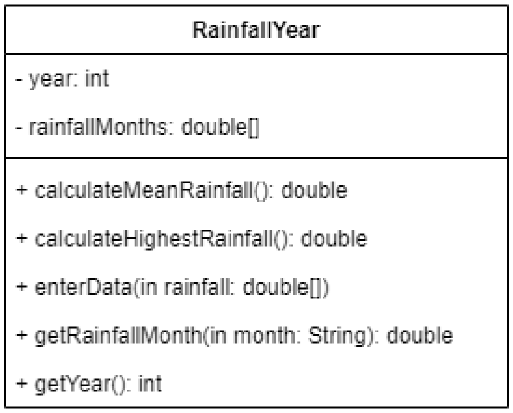
Similarly, change `reference` to "HO12345" and run the program again. Again no match will be found. Try out different strings to see what happens.

Save all your work, and continue with the exercises.

Exercise 1 (2 Marks)

This exercise requires you to create a new class which will make use of arrays and repetition statements.

First, ensure that the code for this exercise is added in the `src/main/java/lab5_exercise1` package. Create a new class called `RainfallYear`. The `RainfallYear` class is used to store the mean monthly rainfall data for just one year using an array of doubles respectively. As well as having a getter for the year, this class also has methods which will allow the mean rainfall for a month to be returned when a month value is specified. Two methods for calculating the mean rainfall and the highest rainfall over the whole year are also provided. The UML diagram below, shows the attributes and behaviour of the `RainfallYear` class:


RainfallYear
- year: int
- rainfallMonths: double[]
+ calculateMeanRainfall(): double
+ calculateHighestRainfall(): double
+ enterData(in rainfall: double[])
+ getRainfallMonth(in month: String): double
+ getYear(): int

The notes below can help you structure your code:

- Declare a new `int` variable to hold the year in question.
- Declare an array field called `rainfallMonths` without defining its size.
- Create a parameterised constructor. Within the constructor, initialise both fields. The array should contain 12 elements of type `double`. Look at the JUnit file to identify the number and type of parameters required for this constructor.
- Do not define any setters and getters for the array. Only a getter for the `year field`
- Define four new methods called `calculateMeanRainfall`, `calculateHighestRainfall`, `enterData` and `getRainfallMonth`.
  - The `calculateMeanRainfall` method should be defined as a function with a `double` return type.
  - The `calculateHighestRainfall` method should also be defined as a function with a `double` return type. The value returned represents the highest rainfall over the year.
  - The `enterData` method should be defined as a procedure, with the responsibility to initialise the field array. The values of the array parsed as a parameter are used to define each element in the `rainfallMonths` array.
  - The `getRainfallMonth` method returns the rainfall of the specified month. A switch case can be used that determines the position within the array using the `month` value.
- Some of the methods might require examples of control flow; repetition and/or conditional statements.
- Test your code by right-clicking the project name and then `Run As > JUnit Test`. JUnit tests have been predefined in the `src/test/java` source folder. Do not modify them as this would cause your Maven Test to fail!

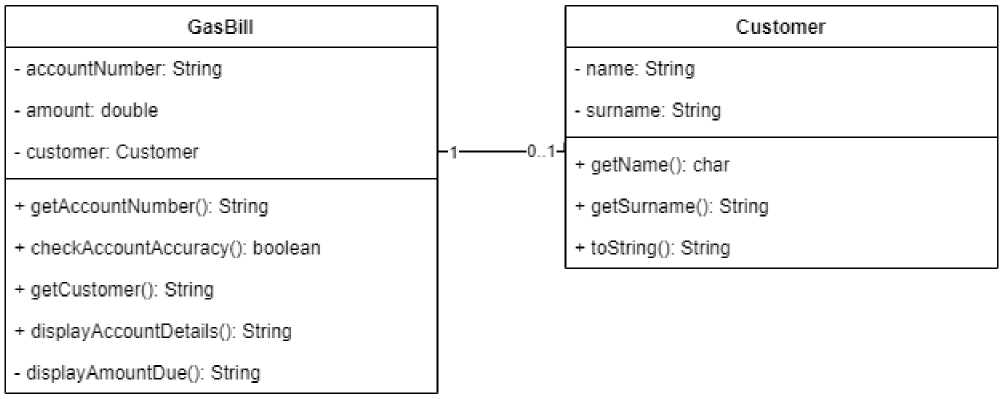
Exercise 2 (3 Marks)

This exercise requires you to create a new program using your understanding of Strings and conditional statements.

The code for this exercise, will need to be added in the `lab5_exercise2` package.

The aim of this exercise is to create a brief note of someone's gas bill, which includes the gas bill account number and the customer's details. A gas bill account number is made up of 12 digits. The program should check that a particular gas bill string is made up of 12 digits, for example `1234-5678-1234` where the three groups of four digits are separated by hyphens.

The `GasBill` and `Customer` classes are related as shown in the following UML class diagram:



- Define a `Customer` class. Each `Customer` object comprises of a name and a surname.
- Within the `Customer` class, define a parameterised constructor with two parameters.
- Define two getters. **Note**: One of the getters returns a character instead of a `String` value. Consider using one of the built-in methods to extract the required letter.
- Define a `GasBill` class. A `GasBill` object contains a `String` value that represents the account number.
- Within the `GasBill` class, define a parameterised constructor with three parameters. Initialise the fields as expected. A gas bill account number is made up of 12 digits. The program should check that a particular gas bill string is made up of 12 digits, for example `1234-5678-1234` where the three groups of four digits are separated by hyphens. If the account number does not match the required format, the `accountNumber` field should be re-initialised with the value:

```
Invalid Account
```

Note: A conditional statement is required to perform this check. Consider adding this in the constructor so that only valid objects can be created.

As per the class diagram, the `GasBill` class also has five methods; two accessor methods, a `checkAccountAccuracy()` function that returns a `boolean` value and two user-defined methods called `displayAccountDetails()` and `displayAmountDue`. *Note: The latter is set to private).*

- Define the methods in the `GasBill` class. Use the defined JUnit test classes to guide you of the expected functionality and output. The `display..()` methods need to return a `String` value without printing the value on the console. The `System.out.println()` statements are not required.

  The `displayAccountDetails` should return the following formatted output:

```
Gas Bill
 Account Number:1234-5678-1234
 Customer:S. Kazamia
 Amount due:70.50
```

  OR

```
Gas Bill
 Account Number:Invalid Account
 Customer:B. Ross
 Amount due:100.00
```

  What should the output of the `displayAmountDue` method be? How can the value `100` be returned as `100.00`? Remind yourself on how we formatted the ClockDisplay output in Lab02.

- Test your code by right-clicking the project name and then `Run As > JUnit Test`. JUnit tests have been predefined in the `src/test/java` source folder. Do not modify them as this would cause your Maven Test to fail!

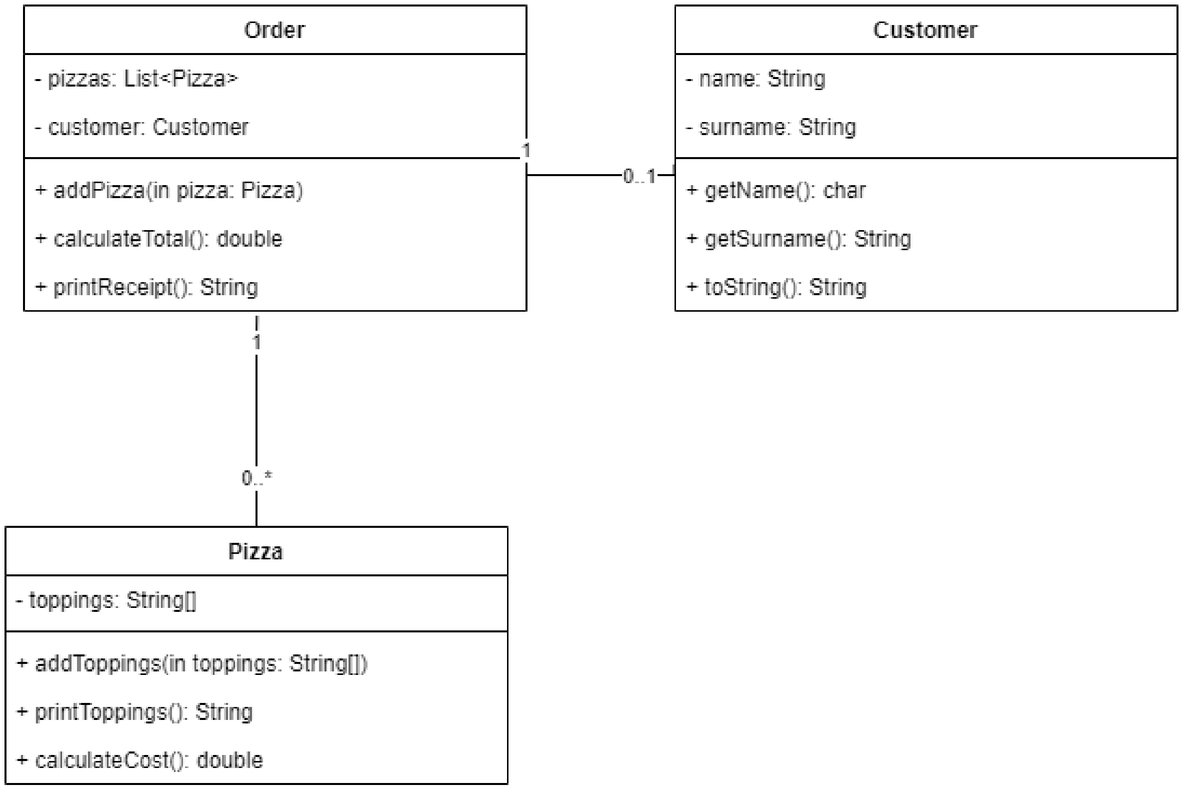
Stretch & Challenge Exercise (5 Marks)

This exercise requires you to use the `Customer` class from the previous exercise.

First, make sure that you use the correct package for your code: `lab5_exercise3`.

The aim of this exercise is to represent a simple pizza delivery system. **Note:** No changes are required in the pre-defined class (`Customer`, but two new classes called `Pizza` and `Order` need to be introduced.

Copy the `Customer` class in the new package, without making any changes to it (except the package name). The UML class diagram below, shows the attributes and methods of the new classes.



- Each `Pizza` object comprises of an array that represents the toppings on the pizza. There can only be 10 toppings on the pizza.
- Define a default constructor for the class that allows an object to be created without any parameters. Initialise the array in the constructor!
- Add the `addToppings()` method. This should be written in a concise manner, by accepting a `toppings` array as input. This method may need a conditional statement and a repetition statement in order to initialise the field `toppings` with the values of the parameter `toppings`. Be careful - make use of the keyword `this` where necessary.
- Add the `printToppings()` method. This should return a String with the required information in the specified format:

```
chicken,cheese,tomato,peppers
```

- Create a `calculateCost` method that sets the cost to the pizza depending on the number of toppings. Use the following incomplete code to guide the development of the method:

```java
//Complete the signature
 () {
          case 1:
          case 2:
          case 3:
          case 4:
               price = 8.99;
               //Add code here
          case 5:
               price = 9.99;
               //Add code here
          case 6:
               price = 10.99;
               //Add code here
          case 7:
               price = 11.99;
               //Add code here
          case 8:
               price = 12.99;
               break;
          case 9:
               price = 13.99;
               break;
          case 10:
               price = 14.99;
               break;
}
```

- You can test your code, by creating a `PizzaTest` class with multiple objects. Try to recreate the above output in your program.
- Once the `Pizza` class is fully functional. You can create the `Order` class.
- Define two fields `customer` and `pizzas`.
- Define a parameterised constructor that only takes a `customer` parameter. Initialise your ArrayList in the constructor.
- Create a new procedure called `addPizza` that accepts a `Pizza` object as its input. The value of the input needs to be added in the pre-defined array list.
- Define a `calculateTotal` method that goes through the array list, and calculates the total cost of all the pizzas.
- Create a function `printReceipt` that formats the output as follows:

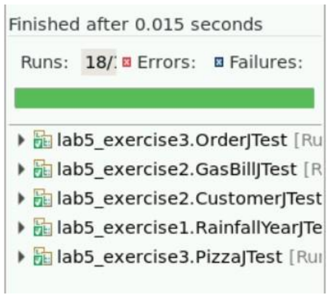
```
Customer: J. Bloggs
Number of Pizzas: 3
Total Cost: 28.97
```

  Do not use a `System.out.println` statement. Instead, return the formatted `String` value.
- Test your code by right-clicking the project name and then `Run As > JUnit Test`. JUnit tests have been predefined in the `src/test/java` source folder. Do not modify them as this would cause your Maven Test to fail!

Final Steps

The final step of the lab activity is to check on whether the code is fully functional. To do this, right-click on the project name and select `Run As > JUnit Test`.

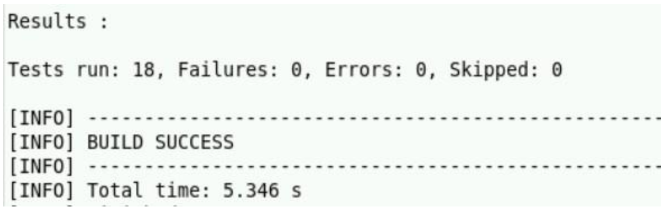
By selecting `Run As > JUnit Test` all the pre-defined JUnit tests will be called. You should get the following in order to proceed:



```
Finished after 0.015 seconds
Runs: 18/  Errors:   Failures:
lab5_exercise3.OrderJTest [Ru
lab5_exercise2.GasBillJTest [R
lab5_exercise2.CustomerJTest
lab5_exercise1.RainfallYearJTe
lab5_exercise3.PizzaJTest [Ru
```

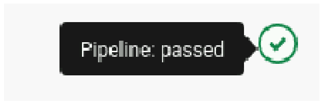
It is now time to test the structure of your Maven project. To do this, right-click on the project name and select `Run As > Maven Test`.

This should return the following message on the console:



```
Results :

Tests run: 18, Failures: 0, Errors: 0, Skipped: 0

[INFO] ----------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------
[INFO] Total time: 5.346 s
```

The test passes - now what?
You can now commit and push all the changes made to the remote repository. To do this, select the `Git Staging` view via the menu `Window > Show View > Other...` and then `Git > Git Staging`. Add all the unstaged changes to your local repository, and commit and push the changes to the remote repository.


Pipeline: passed