JAVA WITH JAVAFX

A SL course for IB

ABOUT THE COURSE

A very practical condensed Java programming course. You will be a Java Programmer quicker than you think. It should be followed in tandem with Topic 3 on my web site. It focuses on one or two ways to accomplish all the common tasks. It is not intended to be comprehensive. The course includes the creation of graphical user interfaces with JavaFX.

P Benson

Table of Contents

Table of Contents

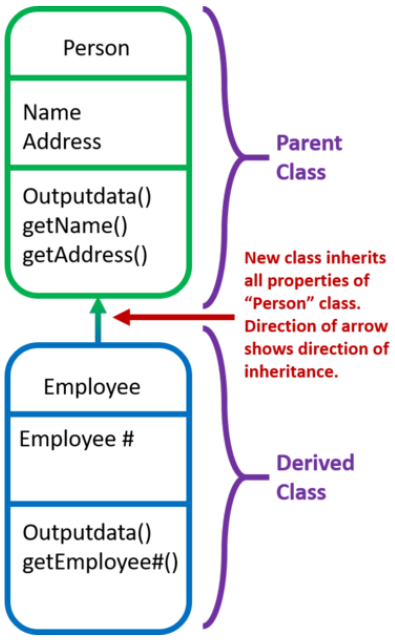
Java is a high level, object orientated paradigm programming language.

Due to its unique away of being both compiled and interpreted it is portable and platform independent from one device to another. Oracle, its developer, maintains that there are 3-4 billion devices running Java. Current statistics show it is still the most popular language.

The Java compiler is called **javac.** It translates .java files into .class files that store the bytes (**bytecode**) and not an executable. The Java interpreter is a Java Virtual Machine (JVM) called **java** which runs the compiled bytecode on your device.

Wherever there is a JVM installed (like your smart phone) then your program will run.

Java is **object orientated**. (Just like the real world). Objects are created **(instantiated**) with **properties** and **methods** (behaviours) based on a template or blueprint called **classes**. If classes can be related, for example – A Dog IS A Pet, you can **avoid duplicating** code as the dog can **inherit** code from the **super or parent** class - Pet. Any pet will have the **properties** of a type or breed and **behaviours like** move, eat and sleep etc just as a dog will. In Java you can only inherit from one other class (There are ways around this though). Classes that inherit from a parent or **base** class can be called **sub**, **derived** or **child** classes. Classes are documented using UML diagrams like this one:



Classes can be **composed** of or contain other classes. For example a car object will have an engine object and many others

- **Everything** in Java is defined in a class.

Introduction to course methodology.

This course works on the basis that to learn Java you must write programs. A lot of technical information will only be given to you as you need it rather than give you lists and tables of methods up front which mean little at the early stages of learning Java. It uses the **Eclipse IDE** which has been given to you on a flash drive.

I would like you to keep the code you write so you can use it as a reference or even as a template. Code up even the small snippets of code you come across.

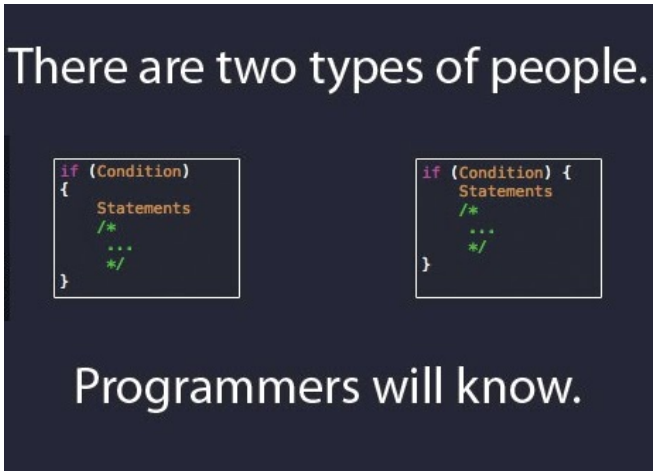
One option is to solely learn **console-based** applications running in a command window. This is all you need for Paper 2 questions as well as Option D. It is also quicker to demonstrate concepts and implement algorithms without the fuss of creating GUI widgets.

This can be slightly boring though, because let's face it we like to see and click on "stuff". A window with input fields, buttons, message boxes, menus and images, thank you very much! For the project in HL your user will probably expect an interface.

JavaFX is an external library that can be used to create graphical user interfaces.

Therefore, we will do both. You will design console-based programs as well as JavaFX GUI applications. Some required skills like layouts will be demonstrated rather than make this document too verbose.

Java uses curly braces { } to encapsulate sections of code like classes, methods, if statements and loops. A source of frustration will be matching these up. In Eclipse clicking next to a bracket will show its partner.



In this course I have usually included my preferred way of coding a solution. There are normally a myriad of other ways to do the same thing which may be more efficient or up to date. Java is enormous and that is to be expected. You will find your own alternatives if the code in this course does not quite meet your needs. Please share these with others. That is a lovely feature of programmers generally – they don't keep good ideas to themselves.

Remember programs rarely work first time. If yours do, then that would be "weird". So be resilient when things go wrong.

// You should "pepper" your code with comments. I promise that you think you will remember what your code does, but after a week or two, you won't.

Forums especially Stack Overflow are useful resources. Google for a solution to a problem and you will normally find a variety of answers. Otherwise ask for help, but not before evidencing you tried your best to solve it first.

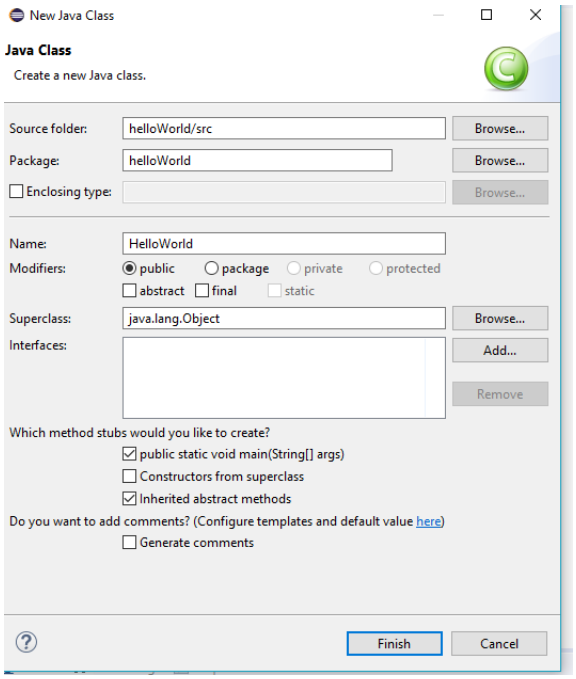
http://www.isel.ufu.br:9000 is a useful search engine for Stack Overflow

Let's get coding!

JAVA console application 1 – Hello World

Open Eclipse and select File > new > java project. Give the project a suitable name helloWorld and a package name or leave as default sample.

Click on the new folder and File > new Class> add the name starting with a capital H and click finish.




```
package com.helloWorld;

/* This is a simple Java program.
   FileName : "HelloWorld.java". */

public class HelloWorld    //every app must have at least 1 class
{
   // Every application begins with a call to main().
   // Prints "Hello, World" to the terminal window.

   public static void main(String args[])  //every app has just 1 main method
   {
      System.out.println("Hello, World");
   }
}
```

After writing the above code click on the run symbol 

Note how you can add multi line comments between /* */ and single line comments with // which the compiler ignores

If you look at the screenshot when we saw the new class window there is a checkbox to add comments ready for you.

---

package com. helloWorld - You can make more than one package and save similar classes in them. You can even have packages within packages to keep your organised. As well as keeping organised Packages avoid same name clashes but still offers ways of sharing data between classes in different packages.

A Java application is a public Java class with a `main()` method.

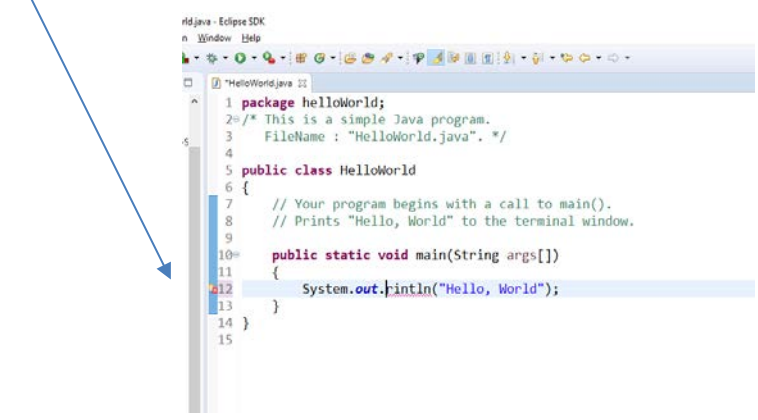
- The `main()` method is the entry point into the application.
- The signature of the method is always: Static because Class Object is null.

```
public static void main(String[] args){
}
```

```
System.out.println("Hello, World");
```

This is how we get output to the console.

If you see a red cross here and a squiggly red line there is a syntax error. You can see a missed out 'p' in print



Introduction to JavaFX

Although you can put all the code, the processing and GUI design in one main class we are going to learn JavaFX by using the **MVC design pattern**. We are going to have an FXML file (**The View**) to design the widgets and **link** this to a controller class **(The Controller**) which does all the processing. For example, what happens when you click on that button? A small main class will set up the window and start the program. The M in MVC stands for **Model** and this is the data class which could be a SQL database for example.

Each view has its own controller.

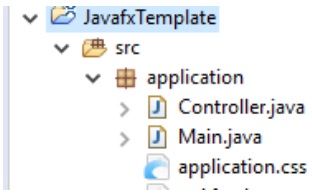
We will use a design interface application called SceneBuilder to populate our FXML file. It is a drag and drop **WYSIWYG** widgets application. You don't need SceneBuilder necessarily as you can design directly yourself using code.

AWT and Swing are other long established libraries to create interfaces. I had to choose one and I like that you can easily add css styles to JavaFX widgets.

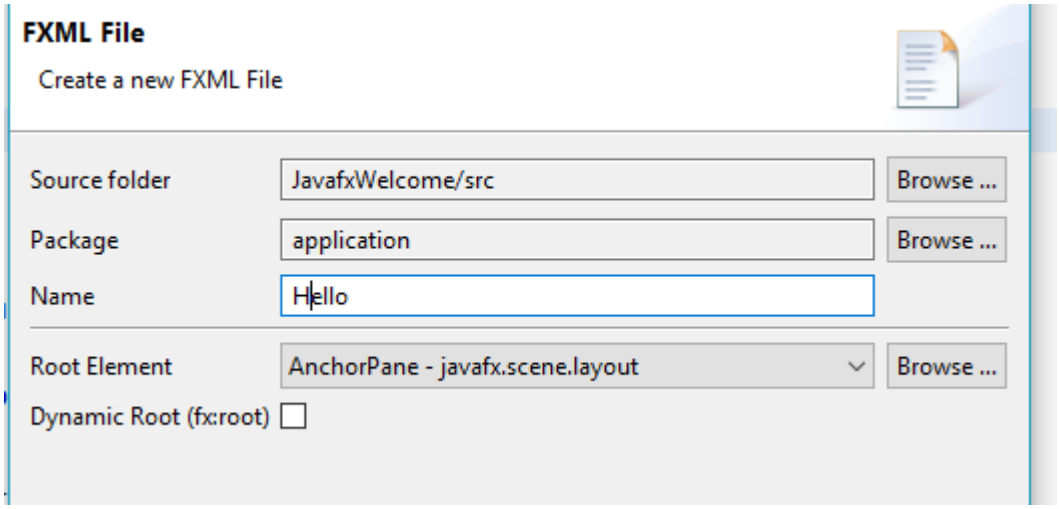
In summary, we still have a Main Class which is connected to the FXML file which in turn is connected to a Controller Class.

JavaFX application 1 – Hello World

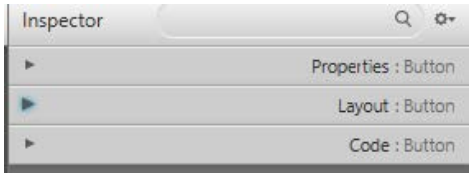
1. In Eclipse's project perspective copy the JavaFx template and rename the copy project
2. Open up the new project in Eclipse's Package Explorer.



3. Right click on the application package> new >other >JavaFX>New FXML Document. Give it a suitable name

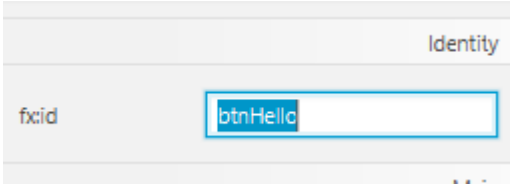


4. Right click on the new Hello.fxml file> Open with SceneBuilder
5. Drag a grid pane from the containers' library with 2 rows and columns
6. Drag and drop 2 labels, 2 TextFields and one button into the grid

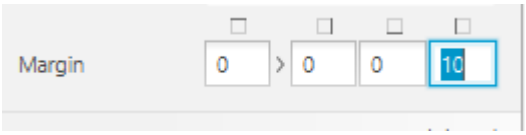
7. There are 3 accordion menus over on the right to set a widget's properties, layout and code the 

8. Give appropriate names to widgets in the code accordion menu

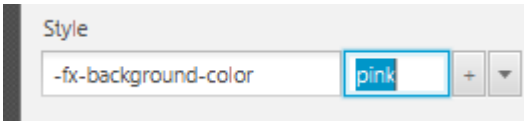
lblName, txtName, lblMessage, btnHello into the appropriate table cells



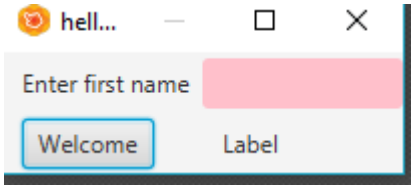
9. **Use appropriate margins in the layout menu**



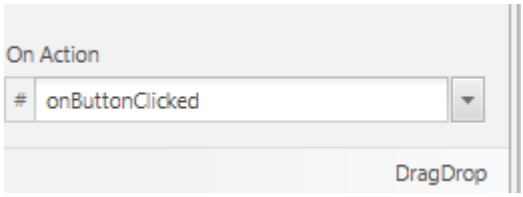
10. In the properties menu you can add colour to the widgets



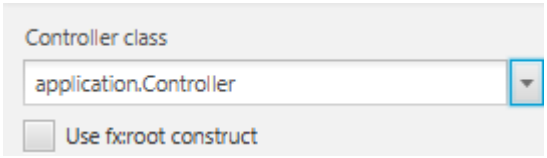
11. Preview the results.



12. Before we leave there are 3 things to do. Click on the button and go to the code menu. Type in or select onButtonClicked. This is how the button knows which code to run in the controller



13. Secondly connect the fxml gui file to the controller. On the bottom left of SceneBuilder



14. Finally, Connect the hello.fxml file to the Main class by changing the name.

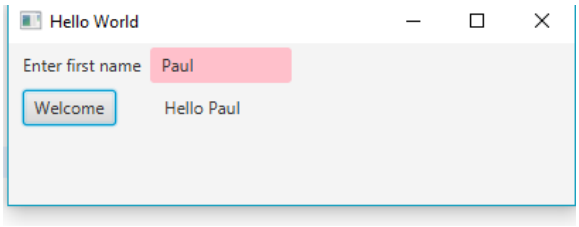
```
Parent root = FXMLLoader.load(getClass().getResource("hello.fxml"));
```

15. Back in Eclipse In the controller.java uncomment the @FXML declarations and add your 4 widget variable declarations names. The same names as you called them in SceneBuilder.

16. In the onClickButton method we will add our code

```
public void onButtonClicked() {

        lblMessage.setText("Hello "+ txtName.getText());

        }
```

Run the code



See how we concatenated the "Hello" with whatever they put in the text field and how we used .getText() and .setText() to retrieve the text and also to set it with a new string.

Some important features in Java

To run Java programs the Operating System needs to know where the Java binary exe files are for the compiler and interpreter. This location is called the **Path or Path Environment Variable. It will be the bin file path for the JDK you installed. There are many different versions of JDK. Go for a long term support version which is versions 8 or 11 currently. Choose either Oracle's OpenJDK version or** Amazon Corretto which focuses only on long term versions (8 or 11)

See this explanation of how to set this up for Oracle in various OS's and here for Amazon. I have provided version 8 and Eclipse on a flash drive.

The Class Path variable is different. The Java JRE needs to know where your classes are to be able to run them. This is important if you have an application with different folders and packages and are running Java from a console.

See the appendices to set up Java's paths to run from command prompts.

If you have trouble in Eclipse try: project > clean or rebuild in intellij

Just be careful to right click on the relevant package to create a new class.

Java is a statically-**typed and** strongly-**typed language.**

In a statically typed language, every variable name is bound both:

- to a type (at compile time, by means of a data declaration)

int time =6 //type integer

- to an object.

Dog fido = new Dog //object Dog

The binding to an object is optional — if a name is not bound to an object, the name is said to be null.

Once a variable name has been bound to a type (that is, declared) it can be bound (via an assignment statement) only to objects of that type; it cannot ever be bound to an object of a different type. An attempt to **bind** the name to an object of the wrong type will raise a **type exception.**

In a **strongly typed language,** variables can NOT be implicitly coerced to *unrelated* types.

a =”9”
b=9
c =a + b would give an exception error

To do this some type of conversion is needed **(casting)**

Java passes parameters by value, in other words a copy of the reference.

The declaration of variables

```
int count
int speed = 30; //Declaring and Initializing integer variable
char b = 'd'
double amount = 0.00
boolean flag = false
String s = "Let us not go lightly”
```

Some variable types like int are primitive and some like Strings are objects which have methods to operate on them.

Variables are called **local variables** in methods. Meaningless outside of the method.

Instance variables are the properties of an object instantiated from a class.

e.g. *dateOfBirth* as part of a student class.

Static variables are part of the class available to all objects of the class but not part of an object's properties. If instance and static variables are not initialised Java will insist on 0 and do it for you.

Boolean data type is used for **logical values**.

Access and Non Access Modifiers

Classes, fields, constructors and methods can have different Java access modifiers. It is all about controlling access to objects and their methods and properties.

We may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used:

private means only that the code inside the class can access. Even sub-classes cannot

public says that any part of the program can read it (otherwise it could only be read by code written in the class it is in. No good for button handlers for example).

public inside a method doesn't make sense. Variables defined inside methods can only be accessed while inside that method, and once the method is complete they are thrown out. So, it would be impossible for those variables to be public

protected gives access to sub classes even and everything in the same package. No other access is available

final is a non access modifier used to declare a constant says that the value of Name will never change once it has been assigned. final in a class definition means the class cannot be extended. **fina**l describes how the variable is going to be used. It makes sense within a method, because it is not about access.

Final local variables must be initialized only once before they are used.
Final method parameters are initialized on the method call.

Constants – They don't change and created by using final keyword

Say you want VAT to always be 20 the you use **final** to set that up

```
public class CalcVats {
   public final int VAT = 20;
}
```

Methods marked as *final* **cannot be overridden.** When we design a class and feel that a method shouldn't be overridden, we can make this method *final*.

static is a non access mofiiersays that the Name variable is a part of the class itself, not of an instance of the class. In other words, all instances of your class share the same Name variable. "There is only one"

static specifies how it is accessed (either on an instance of the class or through the class itself). Inside a method this doesn't make sense, because local variables are discarded after the method closes, so nothing else will be accessing it.

Iteration (loops)

I intend to cover these as and when we need them in the programming tasks. The rule of thumb is to choose a **for loop** if you know how many times you need to iterate.

```
for (int i = 0; i < 6; i++) {
    System.out.println("A basic for loop"): i = " + i);

}
```

and a **while loop** if the number of iterations depends on some condition being true or false.

```
boolean quit  = true;

while (!quit) {

   //code

}
```

You may want at least one iteration come what may

```
boolean quit= false;
do {
    //code
   quit =true

 } while (!exit);
```

Selection (If....)

Most of your code may not be executed as it is conditional on something being true or false.

```
int clock = 22;
if (clock < 10) {
  System.out.println("It's morning radio.");
} else if (clock< 20) {
  System.out.println("Day time tv.");
} else {
  System.out.println("The evening news.");
}
// Outputs "The evening news."
```

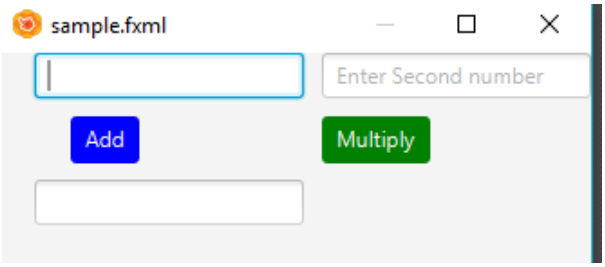
If you have a lot of conditions which are grouped, you should use a Switch statement

```
int day = 2;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
```

```
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Tuesday"
```

JavaFX application 2 – Design a simple add and multiply calculator.

Copy and paste the template renaming it as before. Create a new FXML file and open in SceneBuilder



1. Add the widgets **txtFirstNumber, txtSecondNumber with suitable lables , lblResult and btnAdd and btnMultiply.**
2. In the controller.java uncomment the @FXML declarations and add your widget variable declarations and import the javaFx versions of any required library
3. In the onClickButton method create the code. You will have to know which button was pressed.

```
@FXML
      public void onButtonClicked(ActionEvent e) {
                  if (e.getSource().equals(btnAdd)){
          //code here if this button is clicked
```

4. We will have to cast string values to numbers as adding a string simply concatenates them like “2” + “3” = “23”. Use Use parseInt to do this

```
int one=Integer.parseInt(firstDigitField.getText());
```

same for **int** two =.....

5. We will have to convert the numbers back to a String to display the results. Use valueOf for this

   answer =String.*valueOf*(one+two)

6. Don't forget to connect the Main class to the FXML file and the FXML file to the Controller class.

Java Console application 2 – Swap numbers

Requirement Your program prompts user to enter two numbers Clicking a button should swap them over if the top one is a higher value than the lower one. A message should appear saying swapped or not swapped.

Using the Scanner Class to obtain command line input

```
import java.util.Scanner;

public class work
{
        static Scanner sc =new Scanner(System.in);

        public static void main(String[] args)
        {
                System.out.print( "Enter number : ");

                int i = sc.nextInt();
                System.out.println("You entererd " + i);
        }

  }
```

Remember the method for designing algorithms

1. Identify the variables you will need either to hold input or for processing with their type and initial value.

2. Identify any collections needed eg. array, hashmap , list etc

3. Identify the type of iteration needed if any and how many iterations are required, if known.

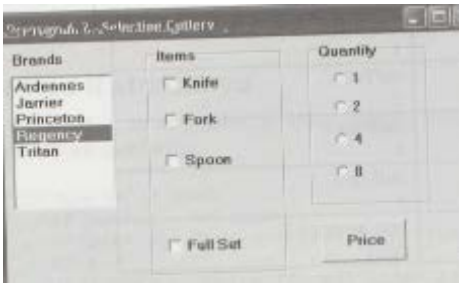
4. Identify the processing required

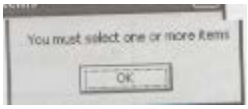
5. What outputs are needed?

Use this system to write pseudocode before writing the program

JavaFX application 3 – Viewlist, Check boxes and radio buttons

Requirement Write a program for a company to sell cutlery. The user should select a manufacture in the view list, knife, fork and spoons in the check boxes and quantity in the radio box. A button will bring up an alert box saying their choice has been added to the basket or if a choice has not been made an appropriate message asking the user to make a selection.





Confirmation Alerts code

```
  button.setOnAction(e ->{
            Alert alertvuescan
 =new Alert(AlertType.CONFIRMATION);
            alert.setTitle("OK to do this");
            alert.setContentText("Click OK to buy");
            Optional <ButtonType> action =alert.showAndWait();

            if(action.get() == ButtonType.OK){
                  System.out.println("oK");
            }
        });
```

ERROR message alert box

```
      Alert alert =new Alert(AlertType.ERROR);
            alert.setTitle("No choice Error");
            alert.setContentText("Please tick a box");

            alert.show();
```

Checkbox

A checkbox is created simply after importing control and declaring the variable

CheckBox box1 = new CheckBox("Tuna")

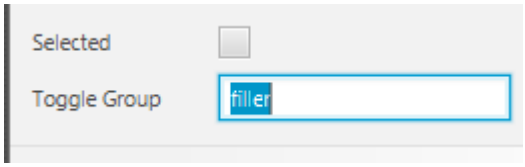
```
public void checkCheck(ActionEvent event ){
		if(box1.isSelected()) {
			System.out.println("yep");
		}}
```

Radio buttons

A simple radio button on its own is straightforward once you have imported the control and declared the variable in the Controller- r1.isSelected

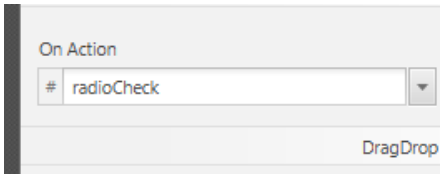
If there are a group of them and only one can be selected you need to enclose them in a toggle group.

Select all the named radio buttons and In SceneBuilders properties add a toggle id in the Toggle Group text field.



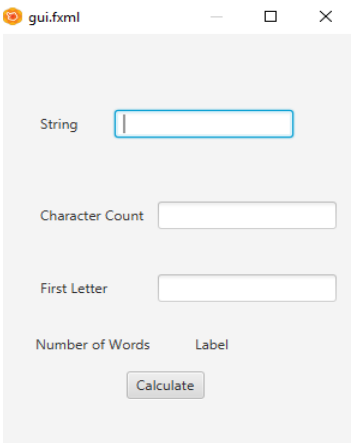

```
@FXML
	private RadioButton rb3;
	@FXML
	private ToggleGroup filler;

	public void radioCheck(ActionEvent event ){
		if(rb1.isSelected()) {
			System.out.println("Cheese");
		}

		if(rb3.isSelected()) {
			System.out.println("Prawns");
		}
		if(rb2.isSelected()) {
			System.out.println("Tuna");
		}}}
```

Select the action method for each button in code menu



JavaFX application 4 - Playing with Strings

Requirement Write a program to enter a word or sentence and then output the number of characters in the string, ignoring spaces before and after. Output the first character as an upper case character and finally output the number of words in the string



Copy and paste the template renaming it as before. Create a new FXML file and open in SceneBuilder.

1. Add the widgets **txtString, txtCharacterCount, txtFirst, lblSentences and btnCalculate to the anchor pane. Use appropriate padding**
2. In the controller.java uncomment the @FXML declarations and add your widget variable declarations and import the javaFx versions of any required library
3. Create the code which returns back a string to lblInterest. You may want to a function for this which the onButtonClicked method calls.
4. Don't forget to connect the Main class to the FXML file and the FXML file to the Controller class.

Common string Operations:

To split into separate words based on a delimiter like spaces or commas

Mercedes Benz, Petrol,Manual

String str[] = stringToSplit.split(",");

String findFirstWord =str[(0)] ;

output: Mercedes Benz //we split on a comma and not a space

Can then use string.length to find how many words

to trim spaces before and after the string

```
String t = txtWord.getText().trim();
txtWord.setText(t);
```

To get number of characters

```
int p = t.length();
```

To get first character as a string and make it uppercase

```
String f = txtWord.getText(0, 1).toUpperCase()
```

To get last character as a character

char lastChar =string.charAt(string.length() - 1); // -1 is last , -2 last but one etc

To find the position of the first occurrence of a character

```
int pr = txtWord.getText().indexOf('s'); /*find index of first ‘s’. Can add an integer .indexOf('s',5) as an optional start point. Returns -1 or false if none there*/
```

To count sentences

```
int counter = 0;
for (int i = 0; i < t.length(); i++) {
     if (t.charAt(i) == ' ') {
                  counter += 1; }
lblWords.setText(String.valueOf(counter + 1)); }
```

To see if a string contains a substring

```
if (string.contains("CPD")) {

}
```

To check if two strings are the same - can't use ==

```
if (string1.equals(string2) ) {
```

```
if(should_be_randomsubstring.equals(randomsubstring)){
    System.out.println("Failure")
}else{
    System.out.println("Success");
}
```

To convert a String to a double or integer in JavaFX textFields

```
double total =Double.parseDouble(txtField);
int total2 =Integer.parseInt(txtField);
```

and back again

```
String txtanswer = Double.toString(total);
```

Concatenating Strings

a = "9"

b = " 5"

a + b will output "95"

(a.concat(b)) will do the same

// Function to concatenate two Strings in Java using StringBuilder

In the Website notes there is a discussion about how Strings are immutable and to change the actual object and not a copy we can use the StringBuilder class

```
public static String concat(String s1, String s2)
{
        return new StringBuilder(s1).append(s2).toString();
}
```

To replace a character with another

a="Tunes"

...println(a.replace('T', 'R')): //outputs Runes

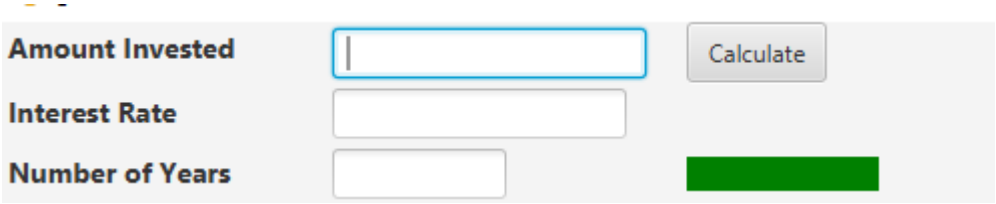
To finds words that start with or end with a substring

//words is an array of strings

```
for(String w : words){
      if (w.startsWith("fu")) {
```

JavaFX application 5 - Design a calculator to calculate compound interest

Requirement Write a program to calculate the amount of interest earned on an investment.
The amount invested, the interest rate and the number of years the investment lasts are to be entered by the user.



The function calculation is **repeated for every year** of the investment:

interest = interest + ((amount invested + interest) + interestRate/100)

The amount invested and the Interest should be doubles and the rest integers

Copy and paste the template renaming it as before. Create a new FXML file and open in SCeneBuilder

1. Add the widgets **txtAmountInvested, txtInterestRate, txtYears,** lblInterest and btnCalcInterest. Use appropriate padding
2. In the controller.java uncomment the @FXML declarations and add your widget variable declarations and import the javaFx versions of any required library
3. In the onClickButton method call the calculateInterest function giving it the 3 parameters
4. Create the calculateInterest function which returns back a string to lblInterest
5. Don't forget to connect the Main class to the FXML file and the FXML file to the Controller class.

Getting Input from the user in the console

The most popular method is by using the scanner class. Try it out.

```
1.  import java.util.Scanner;
2.  class GetIUserInput
3.  public static void main(String args[])
4.  {
5.  Scanner in = new Scanner(System.in);
6.  String s  = in.nextLine();
7.  System.out.println("Checking you entered  "+s);
8.  int i = in.nextInt();
9.  System.out.println("Checking you entered integer "+i);
10. }
11. }
```

Random numbers

It is difficult to make a program **nondeterministic** and do random things which is what you want when designing games for example. The best we can do is **pseudorandom** numbers based on a **seed** which numbers are generated from.

1. Using the Maths class (Math.random)

```
(Math.random() * ((max - min) + 1)) + min //would give a double type random numberth
```

+1 is needed for a random number between two numbers as the max range value is not included.

Create a random integer between 1 and 6

```
public static double getRandIntRange(double min, double max){
    double x = (int)(Math.random()*((max-min)+1))+min;
    return x;
}
System.out.println(getRandIntRange(1,6));
```

2. The method nextInt takes an integer argument, n, and returns a random integer between 0 and n-1.

The following method creates an array of 5 random integers and displays it.

Write the code and test it.

import java.util.Random;

```
public static void main(String[]  args) {

     Random rd = new Random(); // creating Random object
            int[] arr = new int[5]; //array of 5 integers
            for (int i = 0; i < arr.length; i++) {
               arr[i] = rd.nextInt(); // storing random
integers in an array
               System.out.println(arr[i]); // printing
each array element
     }
```

To limit the random numbers, say to 1 to 6 numbers for a dice game

arr[i] = rd.nextInt(6)+1; //normally (7) would give 0 to 6 as the 7 is not included so the +1 onto (6) makes it 1 to 6

Java console application 3 - Lucky Numbers

Requirement Write a program to show random numbers between 1 and 6 in 2 labels. If they are the same values an alert box message says "You won in ? goes"

Methods

A method is a set of commands which allow you to perform a specific operation in a program. In other words, **a method** is a function; something that your class can **do**. In other programming languages, methods are often called "functions" or "procedures", but in Java the word "method" is more common.

If **void** is used then no return value is expected. Without void and you try not to return a value Eclipse will complain. Try and use short but meaningful verb names for your methods - calculatePay for example.

Method parameters

We can pass values called "arguments" to a method when calling it. A method's declaration includes a list of variables which tell us the type and order of

variables that the method can accept. This list is called the "**method parameters**"

Static methods are normally used if you don't need an object instance like just doing a calculation on a method's argument eg public static fact(n) or

```
public static int generateRandom(int multiplier)
{
return  (int) Math.random() * multiplier

}

//You can call this method

 int numb = generateRandom(100):
```

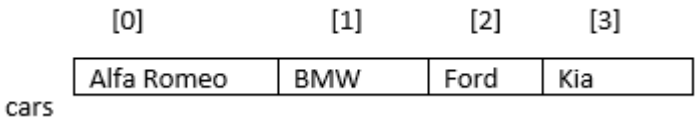
Methods can call other methods. **Try to make a method do only one thing**. If you need it to do 2 things, then create another method.

The main method and any methods in the main method must use the **static** keyword and can be used without having instantiated any class object.

Arrays

Arrays are a vital part of programming as data normally comes in a collection and so we need to spend some time on them. Arrays hold a **fixed non dynamic** amount of data elements as one name and stores in contiguous RAM. Individual indices starting from zero accesses each element. **In Java these elements must be of the same data type.**

1. **To declare an array of 4 cars in RAM**



Write this code in Eclipse

```
String[] cars = {"Alfa Romeo", "BMW", "Ford", "Kia"};
cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Alfa Romeo
```

2. **To declare a new array of 5 empty elements**

```
int[] arrOfInts = new int [5];
arrOfInts[0] =22;

//creates a fixed collection able to contain 5 integers and no more
```

3. **To declare an empty array.**

```
int[] emptyArray = new int[0];
```

Trying to access an empty element will create an arrayIndexOutOfBpound Exception

4. **Iterate** - With any collection structure like arrays we need to know how to iterate through the array accessing each element. We know the index starts at 0. To find the upper bound we can use the .length method.

Traditional method is: add this code in Eclipse

```
for(int i = 0; i< cars.length; i++){
     System.out.println(cars[i]);
}
```

A more efficient method using a for each loop is:

```
for (String strTemp : cars){
System.out.println(strTemp);
}
```

Why is it more efficient? - no counter to increment and it avoids you going out of bounds.

System.out.println (Arrays.toString(car)); // prints out the array but need to import java util arrays

2D arrays

3D arrays are possible too but are less common.

Write this code in Eclipse and save for reference

1. **To declare a 2D array:**

```
int[][] arrayName = new int[2][2]; //creates an 2D array of
2rows and 2 columns
```

2. **To populate a 2D array:**

```
int[][] arrayName = { {2018,2019}
                      {2,6}};
```

---

3. To iterate through a 2D array we would need a nested for loop to traverse the array

```
for (int[] row : arrayName)

    // Loop through all columns of current row
  for (int x : row)
    System.out.print(x + " ");
```

outputs: 2018 2019 2 6

---

4. **To print out in matrix form**

```
import java.util.Arrays;

for (int[] row : arrayName)
        System.out.println(Arrays.toString(row));
```

Outputs: `[2018, 2019]`
`[2, 6]`

Task: create a 2D array to store how many of each car is in stock. Print out the table

Opel	23
BMW	10
Ford	45
Kia	21

Oh dear! Given the description of an array at the start of the section - what is the problem?

To get around the problem you have just identified we can use the strategy below or a collection (See Hash Maps):

1. Create 2 one dimensional arrays as an array of **objects.** It is what java does behind the scenes anyway. A 2D array is an array of an array.

   We need to import the Array Object library for this to work at the top of our program **import** java.util.Arrays;

```
Object[] carsStock[]=new Object[2][];

carsStock[0] = new String[]{"Opel", "BMW", "Ford", "Kia" };
carsStock[1] = new Integer[]{23, 10, 45,21};
```

To print out in matrix form:

```
for (Object[] a : carsStock) {
            for (Object i : a) {
                System.out.print(i + "\t"); //   \t is a tab
            }
            System.out.println("\n"); // \n is a new line
              }
```

```
Opel   BMW    Ford   Kia

23     10     45     21
```

Collections

Collections are more flexible than arrays as they are actually classes with really useful methods allowing adding, deleting, sorting, inserting etc.

Two of the most useful that I find are ArrayLists and Hash Maps

1. Lists – ArrayLists

An array list is a dynamic and ordered array so is more flexible than an array. Putting the contents of a text file in them and process the data from the array list is a common strategy.

Declaring arraylist

```
List<String> testList = new  ArrayList<>();

Int n =5;
ArrayList<Integer> arrli =new ArrayList<Integer>(n) //sets up with an initial size of 5
for (int I =1; i<=n; i++)
```

Find size of arraylist

```
int a= arrli.size();
```

putting contents of a file into the arraylist

```
studentsList =reader.lines().collect(Collectors.toList());
```

printing a particular element

```
System.out.println(testList[0]);
```

sorting an array list

```
Collections.sort(listofcountries);
```

Iterating through an array list

```
for(String temp : testList) {
        if(temp.contains("axe")){
```

2. Hash Maps

Use a totally different dynamic collection called a **HashMap** – and would be the more obvious way of solving the problem with our car stock problem. They can be used for storing usernames and passwords quite easily. ("John", "password1")

To declare a new map

import java.util.Map; at the top of the program

Map<String, Integer> carsStock2 =new HashMap<String, Integer>();

carStock2.put("Opel", 23)

carsStock2.put("BMW", 10) etc

In Java 9 we can do up to 10 pairs like:

```
Map<String, Integer>carsStock2 = Map.of("Opel", 23, "BMW", 10);  etc
```

To get and print the value for Opel

```
int a =carsStock2.get("Opel");

System.out.println("Value for Opel is " + a );
```

```
Value for Opel is 23
```

To print out the whole map

Here is a new way using the forEach method as well as a **Lambda** expression which simplifies the code

carsStock2.forEach((k,v) -> System.out.println("Key = " +K + “, Value = " +v)):

```
Key = Opel, Value = 23
Key = BMW, Value = 10
```

In a Lambda expression you can't change variable values as they act like finals so in that case to print out keys:

```
for (Integer key : map.keySet()) {
    System.out.println(key);
}
```

To print keys and value

```
System.out.println(myMap.toString());
```

```
output {key1=value1, key2=value2}
```

To combine collections -use addAll

```
Collections.addAll(list2, list1) where list 1 will be added to list 2
```

OOP Design

We said at the beginning that Java uses an OOP pattern. As a reminder the advantages of OOP design is that it is:

1. **Modular -** Easy to develop, test and maintain programs
2. **Natural mirror of real-world objects** so easy to understand and develop
3. **Reusable via Inheritance**
4. **Encapsulation** means an object's variables and methods are protected

Procedural programming can do most of this too so if your program is simple just use a procedural design. For a more complex system OOP is the way to go.

Top Tip

To decide if a class can be **derived** from another through **inheritance** see if you can use the class names interchangeably.

"My (dog/pet) will be happy to see me when I get home"

"I flew to Jamaica in a (Jet/Aeroplane)"

"The (weapon/tank) fired at us"

Don't use **inheritance if there is** similar code you are tempted to reuse but you can't use the name interchangeably. It is confusing.

For a sub class to inherit from a parent or **base** class you need to use the `extends` keyword

Use an uppercase first letter for class names and should it be Weapon or Weapons?

Normally use the singular version. To create objects the class definition needs something called a **constructor.** It is a method, obviously public, with the same name as the Class. We can have more than one constructor with different arguments and Java knows which one to use by the number of arguments used when instantiating a new object.

Simple OOP classes using inheritance

Here a tank has most of the attributes of a weapon, so it makes sense for a tank to inherit from a weapon. The tank can still do its own "thing" or a behave in its own way. **Code this.**

```
public class Weapon {

// the Weapon class has three fields
	private int id;
	private int age;
	private String model;

/* the Weapon class has one constructor used to instantiate a new weapon with
default values
	public Weapon() {
		this.id =0;
		this.age =0;
		this.model ="";
		}

/* the Weapon class has another constructor used to instantiate a new weapon of
some sort which you add properties to when you make a new weapon.

	public Weapon(int id, int age, String model) {
		this.id = id;
		this.age = age;
		this.model = model;
		}

// the Weapon class will have getters and setters to allow access to the weapons id
for example

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
		}
	//etc for other properties

 // the Weapon class has one other method for now

	public String fire() {
	     System.out.println("Weapon is fired")}}
```

With this set up we can add a new weapon in 3 ways. We can set up a blank default weapon and add details through the setters. We can set up a new weapon with all the details required included or we can do it later in code.

Properties and methods can be inherited but **not** constructors. Having said that, the parent class fields must be initialised first by a call in addition to the child's.

```
//Tank inherits from weapon but adds its own extra property and fire code

public class Tank extends Weapon {
        private String Commander; //adding a new property just for tank

         public Tank(int id, int age, String model, String Commander ){
              super(id, age, model); //initialises weapon fields
              this.Commander = Commander  //initialises the extra field
      }

          public Tank(){
              super();   /* super is always called anyway. Here it calls
the no parameters constructor to initialise to default values */
          }
         @Override
         public void fire(){ //overwriting the fire() method in Weapon
            System.out.print("Bang!");
      }
}
```

```
public class Main {
      public static void main(String[] args) {
            Tank t1 = new Tank(); //instantiating a tank with null properties
            t1.setId(345);  //setting an id
            t1.fire();  //tank's version
            System.out.println(""+ t1.getId());

            Tank t3 = new Tank(3332,3,"Jaguar", "Bill"); //Tank with properties
            t3.fire();
            System.out.print(""+ t3.getId());
      }
}
```

OUTPUTS

```
Bang!345
Bang!3332
```

You could also do :

Weapon v1 =new Tank(); OR

```
Object obj = new Tank()
```

Then obj is both an Object and a Tank (until such time as obj is assigned another object that is not a Tank). This is called implicit casting.

Normally when developing programs, you wouldn't want to instantiate a Weapon object. Nothing stopping us here though. Later, you will be shown a way where Weapon could not be instantiated.

Java console application 4 - Student Bank Account

Requirement Write a program to show how a Student Bank account class can inherit from a Bank account class. A Bank account will have a customer name, Account Number and balance. Customers can make a deposit or withdrawals. In addition to this a student can have a student loan and the program should return if they have a loan and how much the loan was for.

Polymorphism

The word itself implies "many forms". A Bird class may have a walk method. Some birds will hop, and others walk. This is polymorphism. Overriding methods allow each sub class to do its own thing.

A cool way of using polymorphism is to create an armoury of different weapons in one array.

```
Weapon armoury [ ] = new Weapon[2];   //create an array of type Weapon

armoury[0] = new Tank;      //add to the armoury

armoury{1] = new Machine Gun;   //Add a different weapon to the armoury

for (int i =0, I <2, i++) {

      armoury[i].fire();

}
```

Exceptions

From time to time Java will take exception to errors that you make. It will **throw** this **exception** object at you. Someone will need to **catch** it. Well not literally but the method that caused it, or the method that called that method.

This could be because you tried to divide by zero, read from a file that does not exist or a loop that never ends for example. These exceptions have useful descriptive names like **FileNotFoundException** , **ArrayIndexOutOfBoundsException**, **IllegalArgementException**

If every method "ducks" then the program crashes or does not even compile. These last ones are called **checked exceptions** and must be dealt with. No choice. We can do this using a **try - catch** clause or specify a **throw back** clause which means the calling method deals with the exception. Whichever way we deal with it, a lot of our code will be trying to keep our application running without crashing.

We catch an exception either in the called or calling method by using try -catch

try
code that could produce the error

catch
The system will automatically run the catch instead of its own catch
which would cause the program to crash

finally
Optional code to run whether or not an error occurs E.g. to close a file

Unchecked exceptions usually occur due to human error, where a user tries to input wrong data types for example. If not prepared for they will cause a run time exception

It is probably best practice to prevent the exception handling in the first place

```
if(i >= employeeArray.length) {

    System.out.println("Index is too high!");

     return null;
```

Here is a neat way of enforcing validation when an object is instantiated by the constructor

```
public Volunteer(LocalDate birthday){

       setBirthday(birthday);

public void setBirthday(LocalDate birthday) {
             int age =Period.between(birthday, LocalDate.now()).getYears();
             if (age >=14 && age <100)
                    this.birthday = birthday
             else
```

```
throw new IllegalArgumentException("Sorry wrong age"); }
```

Looking again at our scanner class, here a method passes the exception handling to another method which uses try -catch

```
        static Scanner sc =new Scanner(System.in);

        public static void main(String[] args)
        {
                System.out.print( "Enter number : ");
                int i = GetAnInteger();
                System.out.println("You entererd " + i);
        }

        private static int GetAnInteger() {
                while (true) {
                        try
                        {
                                return sc.nextInt();

                        }
                        catch (InputMismatchException e)
                        {

                                sc.next();// we need this as the while loop will keep checking the
//same input and getting the same error
                                System.out.print( "Not a number : ");
                        }
          }}}
```

Here is a FileNotFound catch

```
public static void main(String[] args){
   readFile();
}

public static void readFile() {
  try {
    Scanner input = new Scanner(new File("file.txt"));
    ...
  } catch (FileNotFoundException ex) {
    // Error message
  } catch (Exception ex) {
    // Incase a different exception is caught
  }
}
```

Your program will work if your catch clause is empty. This is called **"swallowing". Usually done when programmers say to themselves "I will get back to it later"**

Throwback clause example -checked exception

Think of the throws keyword as a promise; You're saying you're not going to catch the exception now, but you're going to catch it at the calling statement.

```
public class FileNotFound {

      public static void main(String args[]) throws IOException {
```

In a main method you have to catch not throw as there is nothing else to catch it in the chain.

Here is a throw with the expectation that the calling method will deal with it (Java knows this as there is no try catch here)

```
public void share(int numberOfApples, int numberToShareWith)
    throws BadNumberException{
        if(numberToShareWith == 0){
            throw new BadNumberException("Cannot divide by 0");
        }

      return numberOfApples / numberToSharewith;
    }
```

Note that if NumberToShareWith is a zero the division does not take place and the calling method has now to catch the exception using a try -catch when it calls the share method.

File Handling

Files could be text or objects. There are a number of ways to **read** and **write** to files in Java. You can use the newer buffered **nio path** classes or the older **stream** based **io file** class with its **FileReader** and **FileWriter** classes. If you have ever streamed a movie and had to wait until the buffer has enough data to start playing the movie you know exactly what a buffer does.

We will use the **io** class but with the BufferedReader and BufferedWriter classes which takes the stream of characters and buffers them in order. This will make things more efficient.

The downside is that this way requires you to deal with Exceptions.

FileReader on its own:

```
FileReader reader =new FileReader("filepath"));
```

With BuffererdReader - includes a FileReader as an argument:

try (BufferedReader reader =new BufferedReader(new FileReader("*filepath*");

The try is a **try with** which saves you having to close the buffered Reader and thus getting an exception. You must add throws IOException to the method header otherwise you would need a try catch: try { BufferedReader reader...etc} and at the end

```
}    catch (IOException e) {
// TODO Auto-generated catch block
            e.printStackTrace();
}
```

Reading a file

I want to reiterate that there are numerous strategies to use io to read and write files. The code below is what I tend to use. However other programmers would do it another way and argue theirs is better.

A really useful strategy is to put the text from a file into an arraylist or other collection and then process the text in the list.
You need testList =reader.lines().collect(Collectors.toList()); **for that after first declaring the testList of course**

You need to watch out for FileNotFoundException which you will get if the file does not exist yet. It is vital to close the buffered reader or stream after using them. reader .close(); Adding try before the instantiating a new BufferedReader will mean Java will do this for you automatically as mentioned above

Reading a file and printing

```
public class Main {
      public static void main(String[] args) throws IOException{

            try (BufferedReader reader = new BufferedReader(new
                  FileReader("src/readIO/shakespeare.txt"))){


//prints character by character using read method


                  int data =reader.read();
                  while(data != -1){
                        if (data == 'H') {
                              System.out.println("Yeah H");
                        }

                        System.out.print((char) data);
                        data = reader.read();
```

```
//Into a String using StringBuilder character by character
            StringBuilder content = new StringBuilder();
                        int value;
                        while ((value = reader.read()) !=-1){
                              content.append((char)value);
                        }

                        System.out.println(content);

//puts text straight into a list
            List<String> list = new ArrayList<>();
            list = reader.lines().collect(Collectors.toList());
            System.out.print(list);

                  }}}}
```

Writing to a file

If the file does not exist unlike reading Java will create one for you.

Creating a text file with some text and the adding to it using write and append methods

```
public class Main {
      public static void main(String[] args) throws IOException {

      //note how similar to the reader version
            try (BufferedWriter writer = new BufferedWriter(new
              FileWriter("hello.txt"))){

             String s ="Hello";
             writer.write(s);
             //now you should open the new text file to test it worked


              //let's append some more text
             writer.append(" World");   // should see "Hello World"
        // for a new line

             writer.newLine();   //give me a new line ready for the next

         }}}
```

Saving the text in an arraylist element by element on a new line using a try catch

```
try {BufferedWriter writer = new BufferedWriter(new FileWriter(new
File("src/application/test.txt"));

        for(int i=0;i<list.size();i++){

             writer.write(list.get(i));
```

```
            writer.newLine();
           }
           writer.close();

           } catch (IOException e) {
           e.printStackTrace(); }
           }
```

Making an object **persistent** by saving to a file

You are designing a war game and want to save your tank objects to use another day. To do this we need to **serialize/deserialize** the object into a stream of byte data.

To serialize we use both the ObjectInputStream (to read) and ObjectOutputStream (to write) classes

1. Add implements Serializable to our class definition

   public class Tank implements Serializable {

   Note any other classes the class references like composition have to individually also implement serializable.

```
package com.objectspersist;

import java.io.Serializable;

public class Tank implements Serializable {

	private static final long serialVersionUID = 1L;
		// Eclipse generated  It is used to verify that the saved and loaded objects have the same attribute


	private int id;
	private int age;
	private String model;

	Tank(int id, int age, String model) {
		this.id = id;
		this.age = age;
		this.model = model;
	}

	@Override
	public String toString() {
		return "Id:" + id + "\nAge: " + age + "\nModel: " + model;
	}
}
```

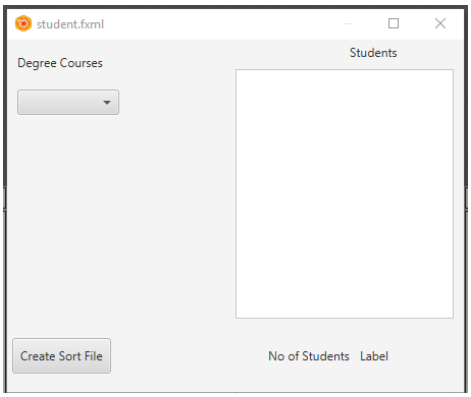
2. Create a class to create, write and the read back the tank objects

```
package com.objectspersist;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;



public class WriteReader {

	public static void main(String[] args) {

		Tank t1 = new Tank(1234, 5, "Tiger");
		Tank t2 = new Tank(2345, 1, "Panther");

		try {
			FileOutputStream f = new FileOutputStream(new
File("myObjects.txt"));
		ObjectOutputStream o = new ObjectOutputStream(f);

			// Write objects to file
			o.writeObject(t1);
			o.writeObject(t2);


		//remember to close
			o.close();
			f.close();

		// read back

			FileInputStream fi = new FileInputStream(new
File("myObjects.txt"));
			ObjectInputStream oi = new ObjectInputStream(fi);

		// Read objects
			Tank tr1 = (Tank) oi.readObject();
			Tank tr2 = (Tank) oi.readObject();

			System.out.println(tr1.toString());
			System.out.println(tr2.toString());

//remember to close.flush forces out any leftovers in the buffered stream!
			o.flush();
			o.close();
			f.close();

		//dealing with exceptions
```

```
            } catch (FileNotFoundException e) {
                System.out.println("File not found");
            } catch (IOException e) {
                System.out.println("Error initializing stream");
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

      }

}
```

JavaFX application 5 – Student Course App

Requirement Write a program which displays the names of students on a selected degree course by reading from a text file - DegreeStudents.txt. A choice box will allow you to select the course. The students on that course will be listed in the List view. Create a label which displays the number of students on the course and then add a button which will create a new sorted file of just the students names..



1. Create the widgets in Scenebuilder
2.
```
public class Controller implements Initializable  {  //need this to
//initialise boxes
@FXML
  private ChoiceBox<String> chDepartments;

@Override //initialise the values in the choicebox
      public void initialize(URL arg0, ResourceBundle arg1) {
            chDepartments.getItems().add("Computing");
            chDepartments.getItems().add("French");
            chDepartments.getItems().add("All courses");
            chDepartments.setValue("All courses"); //set default value
```

```
            //etc.
```

```
 You know need to add a listener to do something when one of the values is
chosen:
chDepartments.getSelectionModel().selectedItemProperty().addListener((v,
 newValue,oldValue) -> select(newValue));
```

3. Open file and read contents into an array list – Use try - catch as cannot use throws exception in an initializer
4. Have a hashmap of dept keys Computing with values COMP -page 30
5. Split the string to isolate the names – page 20
6. iterate through the file putting matched dept code into the list view incrementing a counter as you go. page 37
7. For the new file look to sort an arraylist of all the names and write them to a file -page 30 & 37

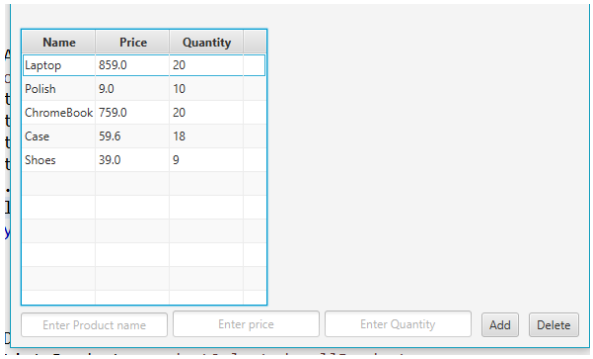
Data Handling

JavaFX includes a TableView which lays data out like a spreadsheet. The data must become an object and can come from a Class object or JSON file for example.

To do this we create our class where the data will come from and populate the table columns with the instance values from instantiated products. To do this the data must go via an **observable array list**. It is easier to show with an example.

JavaFX application 6 – Product Stock List

Requirement Write a program which displays the name, price and quantity of items in stock in a table. There should be a way of adding and deleting stock via text fields and buttons.



1. Copy the JavaFX template and rename. Copy the Products.java file into your Src folder
2. In Scenebuilder set up the form as above adding a table vie, 3 textfields and 2 buttons.
3. In the controller add the widget declarations and import all needed libraries

```
    //set up tableview and column variables
    @FXML private TableView<Product> table; //Called this in Scenebuilder fx:id
    @FXML private TableColumn<Product,String> nameColumn  ; //SceneBuilder go
to the columns fx:id in code and select these names in the drop down
    @FXML private TableColumn<Product,Double> priceColumn  ;
     @FXML private TableColumn<Product,Integer> quantityColumn;
```

4. Now we need to link the table columns with the fields in the Products class

```
@Override
    public void initialize(URL location, ResourceBundle resources) {

            nameColumn.setCellValueFactory(new PropertyValueFactory<Product,
                                                    String>("name"));
           priceColumn.setCellValueFactory(new PropertyValueFactory<Product,
                                                    Double>("price"));
           quantityColumn.setCellValueFactory(newPropertyValueFactory<Product,
                                                    Integer>("quantity"));
           table.setItems(getProduct()); //call the method below to populate
    the table
```

5. Now use an Observable List to create the product objects and populate the table view

```
public ObservableList<Product> getProduct(){
    // This is an arraylist of observable objects
    ObservableList<Product> products = FXCollections.observableArrayList();
    products.add(new Product("Laptop",859.00,20));
    products.add(new Product("Polish",9.00,10));
    products.add(new Product("ChromeBook",759.00,20));
    products.add(new Product("Case",59.60,18));
    products.add(new Product("Shoes",39.00,9));
    return products;
```

6. Now create the add product and delete method – don't forget to link the buttons to these in SceneBuilder

```
public void onAddButtonClicked() {
		Product product =new Product();
		product.setName(txtProduct.getText());
		product.setPrice(Double.parseDouble(txtPrice.getText()));
		product.setQuantity(Integer.parseInt(txtQuantity.getText()));
		table.getItems().add(product);//gets above for table columns
		txtProduct.clear();
		txtPrice.clear();
		txtQuantity.clear();


		}

	public void onDeleteButtonClicked() {
		ObservableList<Product> productSelected, allProducts;
		allProducts =table.getItems();
		productSelected=table.getSelectionModel().getSelectedItems();
		productSelected.forEach(allProducts::remove);
	}
```

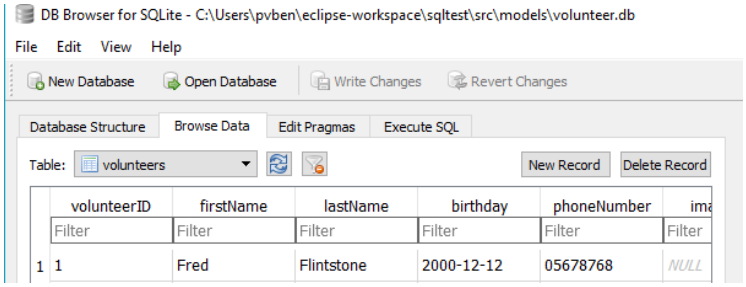
To make this persistent note that Observable lists are not serializable . A work around would be to copy the elements into a normal ArrayList and serialize that. To recover the data from the file copy from an Arraylist back into an observable list to repopulate the Table View.

Connecting to a SQL Database

The standard way of course is to use a database i.e mySQL or Sqlite with the JDBC libraries.

We will connect to a Sqlite database which is a non server based, portable and locally saved version and create a simple log in app.

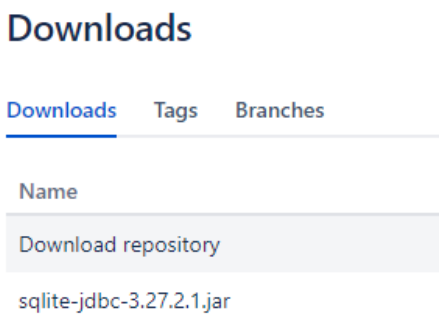
SQLite databases can handily be created using a free program called DB Browser for SQLite which is included on the flash drive. Save the database in the model package in your project



In Eclipse create a JavaFX project with extra packages model, view and resources

To connect to your database:

We need to download the SqLite JDBC connector file from https://bitbucket.org/xerial/sqlite-jdbc/downloads/



Right click on your project > build path > build path >libraries>external jar and find your download

Below would be a typical way of connecting to a database using a class for the purpose. If the database does not exist, one will be created.

```
package application;
import java.sql.*;
public class SqliteConnection {
  public static Connection Connector() {
    try {
        Class.forName("org.sqlite.JDBC");
        Connection conn =DriverManager.getConnection("jdbc:sqlite:EmployeeDb.sqlite");
        return conn;
    } catch (Exception e) {
        return null;
        // TODO: handle exception
    }
}
}
```

Below is a more sophisticated version where a class has a method within it to make objects persistent in a sqlite database.

```
public void insertIntoDB() throws SQLException {
             Connection conn= null;
             PreparedStatement preparedStatement =null; //to avoid sql injection
             try {

                    //1. Connect to the database
                    conn
=DriverManager.getConnection("jdbc:sqlite:src\\models\\volunteer.db");
                    //2 Give a time limit to the query
                 Statement statement = conn.createStatement();
               statement.setQueryTimeout(30);  // set timeout to 30 sec.
                    //3. Create a string the holds the query with ? as place
holders
                    String sql = "INSERT INTO
volunteers(firstName,lastName,phoneNumber)"
                                 + "VALUES (?,?,?,?,?)";
```

```
                    //4. prepare the query
                    preparedStatement = conn.prepareStatement(sql);


                  //5.Convert birthday to sql date
                    Date db =Date.valueOf(birthday);
                    preparedStatement.setString(1, firstName);
                    preparedStatement.setString(2, lastName);
                    preparedStatement.setString(3, phoneNumber);
                    preparedStatement.setDate(4, db);
                    preparedStatement.setString(5, imageFile.getName());
                    //5. Execute the statement. In volunteer app
                    preparedStatement.executeUpdate();


            } catch (Exception e) {
                    System.err.println(e.getMessage());
                          }
        finally {
            if (preparedStatement !=null)
                    preparedStatement.close();
            if (conn !=null)
                          conn.close();

        }

    }
```

A main method will instantiate a new object and then call the insertIntoDB method to save that object.

APPENDICIES

Setting up JAVA to run in a command prompt

1. Open Advanced System Settings

Download and install Version 1.8 or 1.11 of the Java JDK as these are the long term versions.

In Windows 10 press Windows key + Pause Key, This will open the System Settings window. Go to `Change settings` and select the `Advanced` tab.

2. Set JAVA_HOME Environment variable

In “System Properties window” click “Environment Variables…”

Under “System variables” click the “New…” button and enter JAVA_HOME as “Variable name” and the path to your Java JDK directory under “Variable value”

This JDK will be in your program files\Java in your "C" Drive

3. Update System PATH
1. In “Environment Variables” window under “System variables” select Path

2. Click on “Edit…”

3. In “Edit environment variable” window click “New”

4. Type in `%JAVA_HOME%\bin`

4. Test
Open a new command prompt and type in:

```
echo %JAVA_HOME%
```

this will print out the directory JAVA_HOME points to or empty line if the environment variable is not set correctly

Now type in:

```
javac -version
```

this will print out the version of the java compiler if the Path variable is set correctly or *“javac is not recognized as an internal or external command…”* otherwise

You can now run a java program created in notepad say, call it sample.java and in a Windows command prompt run it by typing *javac sample.java*

Some Java/JavaFX errors and solutions

1. `error: reached end of file while parsing`

possible solutions

`missing { brackets for class or bracket facing the wrong way`

In Windows go to Advanced system settings and click on environment variables

Click new and JAVA_HOME and path to jdk underneath

2. **Could not find or load main class**

possible solutions

Update System PATH

1. In “Environment Variables” window under “System variables” select Path

2. Click on “Edit…”

3. In “Edit environment variable” window click “New”

4. Type in `%JAVA_HOME%\bin`

You can also set classpath in windows by using DOS command like:

```
set CLASSPATH=%CLASSPATH%;JAVA_HOME\bin;
```

Classpath error – spellings ok Try project > clean and project rebuild in Eclipse or rebuild intellij

Have you moved a project from one IDE/PC to another and used absolute path addressing. If so swich to relative using the project folders src/…

Javafx errors

3. **Java Error - Illegal Modifier for Parameter - Only final Permitted**

possible solutions

If you haven’t typed void in a method then Eclipse will nag and offer to add a return statement.

`Look at your curly brackets {. Is your code inside another method because of a missing }`

4.the method in type not applicable for the arguments

possible solutions

Either you are not passing a parameter that you should be or the parameter type is wrong. In the method declaration it should be String not TextField for example).

5. Error:Void is an invalid data type for the variable String in method declaration

possible solutions

If it's void don't include String as not returning anything.

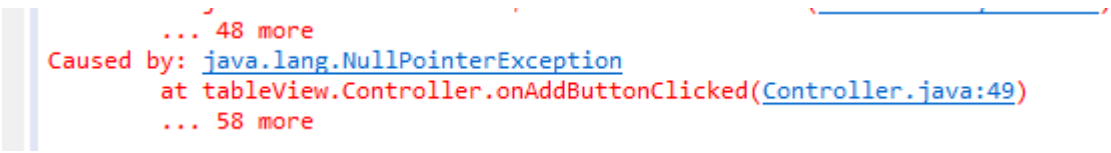
6. Jul 31, 2019 3:18:04 PM javafx.fxml.FXMLLoader$ValueElement processValue

If nothing happens to text field etc

possible solutions

Have you connected the button event handler?

7.



This is very common in Java and JavaFX

possible solutions

If you click on Controller.java.49 it will go to the error in your code

mismatch of widget name between fxml and in controller:

The code below developed this error. txtInterestrate was NULL because the actual name was txtInterestRate in the fxml file but declared as txtInteresrrate in the controller which obviously doesn't have a value

```
ublic void onButtonClicked() {

    //if (e.getSource().equals(btnCalulateInterest)) {

        String answer;
        System.out.println(txtAmountInvested.getText());

        answer=calcInterest(txtAmountInvested.getText(), txtInterestrate.getText(), txtYears.getText());
        System.out.println("Got here too");
        lblInterest.setText(answer);
```

```
product.setPrice(Double.parseDouble(txtPrice.getText()));
```

```
29      private TextField txtPrice;
```

```
See the yellow triangle and exclamation mark which says txtPrice is unused
even though it is. Solution was to look at variable declaration
```

```
7       @FXML
8       private TextField txtProduct;
9       private TextField txtPrice;
0       @FXML
```

```
The name of the field is correct but spot that @FXML is missing
```

Java hates empty objects so thats why main method has to be static as there is not yet an object created. Maybe you named a widget wrong

The best way to avoid this type of exception is to always check for null when you did not create the object yourself." If the caller passes null, but null is not a valid argument for the method, then it's correct to throw the exception back at the caller because it's the caller's fault. *Silently ignoring invalid input and doing nothing in the method* is extremely poor advice because it hides the problem.

```
7. WARNING: Loading FXML document with JavaFX API of version 11.0.1 by JavaFX
   runtime of version 8.0.221
```

possible solutions

Change to this in fxml file

```
xmlns="http://javafx.com/javafx/8.0.0"
```

8. If preview layout is different from Windows

In SceneBuilder, set the Max Height and Max Width to `USE_PREF_SIZE`. Likewise, you can also set the min values.

If you are using Scene Builder, you will see at the right an accordion panel which normally has got three options ("Properties", "Layout" and "Code"). In the second one ("Layout"), you will see an option called "[parent layout] Constraints" (in your case "AnchorPane Constrainsts").

You should put "0" in the four sides of the element wich represents the parent layout.

primaryStage.setMaximized(false);// keeps stage window same size as preview in scenebuilder

9. Illegal modifier for parameter initialise; only final is permitted

Probably forgot to put closing } bracket so a method is part of another.

10. how to resolve "*local variable defined in an enclosing scope must be final or effectively final*" error while trying to write a lambda expression in Java.

problem is: A lambda expression may only use local variable whose value doesn't change. That restriction is referred as "**variable capture**" which is described as; *lambda expression capture values, not variables*. The local variables that a lambda expression may use are known as "**effectively final**".

possible solutions

So how do you use a count variable for example:

```
private static String caser(String name) {
if(name.startsWith(“a”)) {
return name.toUpperCase();
} else {
return name.toLowerCase();
}
}

List<String> names = Arrays.asList(“abc”, “def”, “ghi”);

names.replaceAll(name->caser(name));

System.out.println(names);
```

11. Syntax error on token ";", { expected after this token

Probably misplaced { causing the following i.e code o/s of a method but directly in class

In java you can't simply code loops and other actions as part of the class definition, but rather as method/constructor/block definitions inside the class

13. multiple markers at this line
- Syntax error, insert ";" to complete MethodDeclaration

- Syntax error, insert ")" to complete MethodDeclaration

possible solutions

A bracket of some type is there, and it should not be

actionEvent event() event is not a method so ActionEvent event(

12." java.lang.RuntimeException: java.lang.reflect.InvocationTargetException
at javafx.fxml.FXMLLoader$MethodHandler.invoke(FXMLLoader.java:1774)