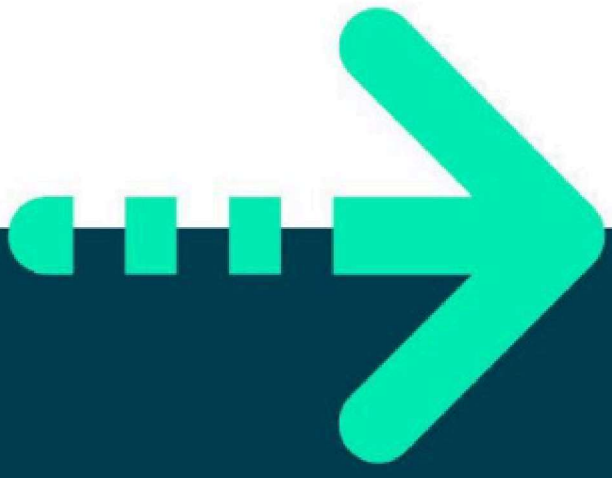
Java Fundamentals

Exercise Guide



CONTENTS

Lab 0 Getting started

The objective of this exercise is to explore the development environment installed on our machines and create an IntelliJ project.

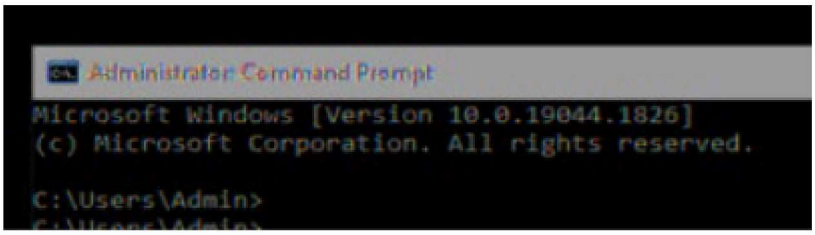
Exercise 0-1 – Explore the environment

Task 1 – Verify the command line Java version

1. To check that Java is available, we use the command line terminal. Press the Windows Start button and type:

   cmd

   followed by Enter to open a terminal window.



2. Check the main Java Virtual Machine version by typing:

   java -version

   This will output the current java version number.

```
C:\Users\Admin>java -version
openjdk version "17" 2021-09-14
OpenJDK Runtime Environment (build 17+35-2724)
OpenJDK 64-Bit Server VM (build 17+35-2724, mixed mode, sharing)
```

3. Check the Java compiler version:

   javac -version

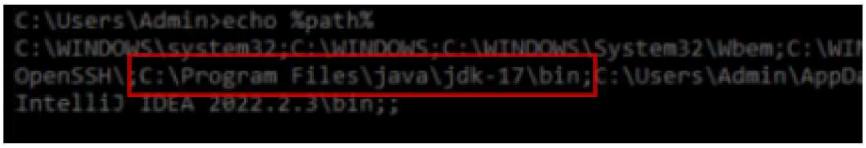
   This confirms that the compiler can successfully be accessed from the command line and that the bin directory path for the **javac** program has been added to the PATH environment variable.

```
C:\Users\Admin>javac -version
javac 17
```

4. Check your current value for the PATH environment variable by typing:

   echo %path%

Your output should look like this:

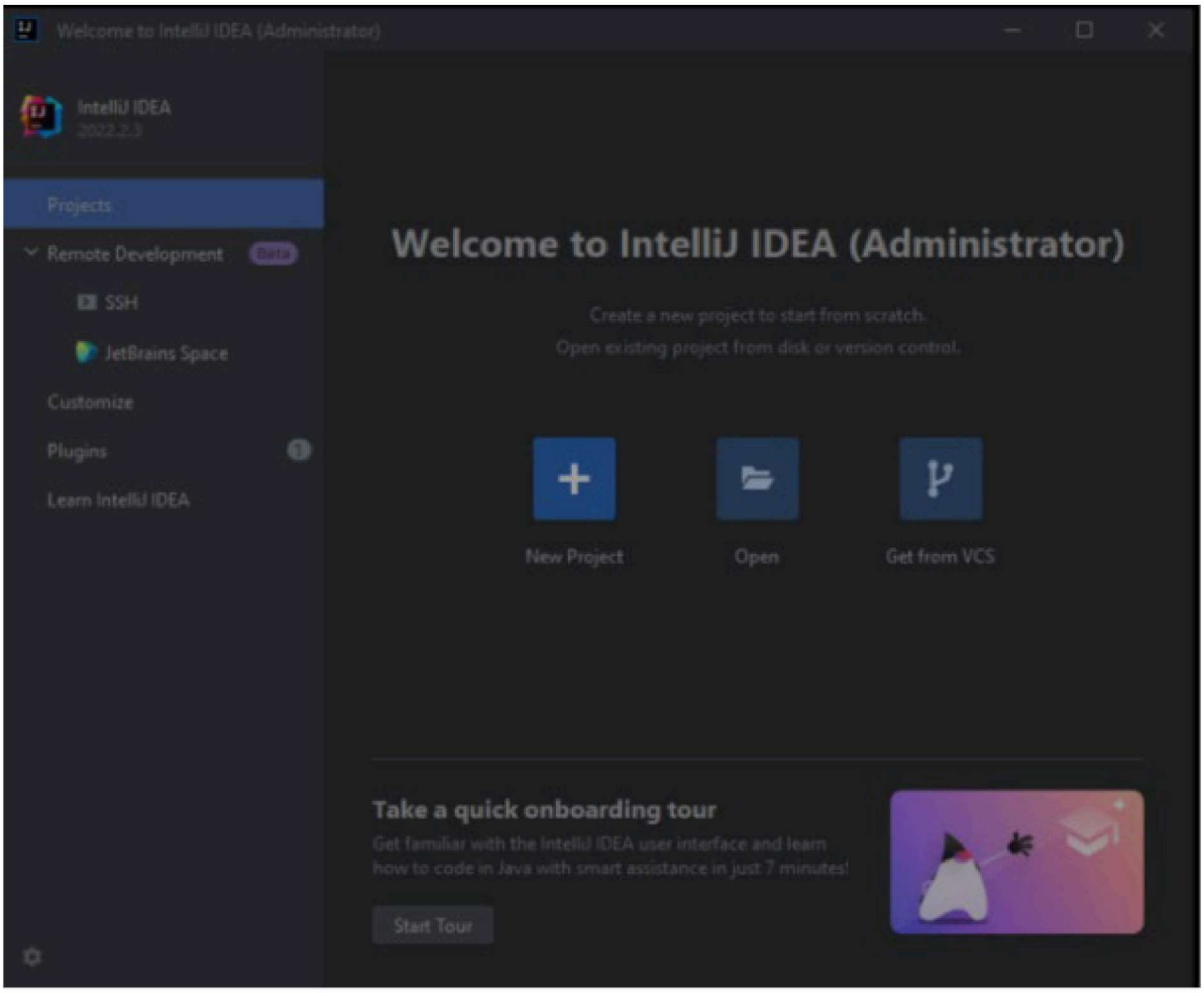


Exercise 0-2 – Introducing IntelliJ

Task 1 – Open IntelliJ and take the Onboarding Tour

1. Click on the shortcut on the desktop to open IntelliJ IDEA:

It will display the Welcome screen:



2. Click Start Tour to take the IntelliJ onboarding tour to get a feel for the IDE. Close the sample project at the end to return to the Welcome screen.

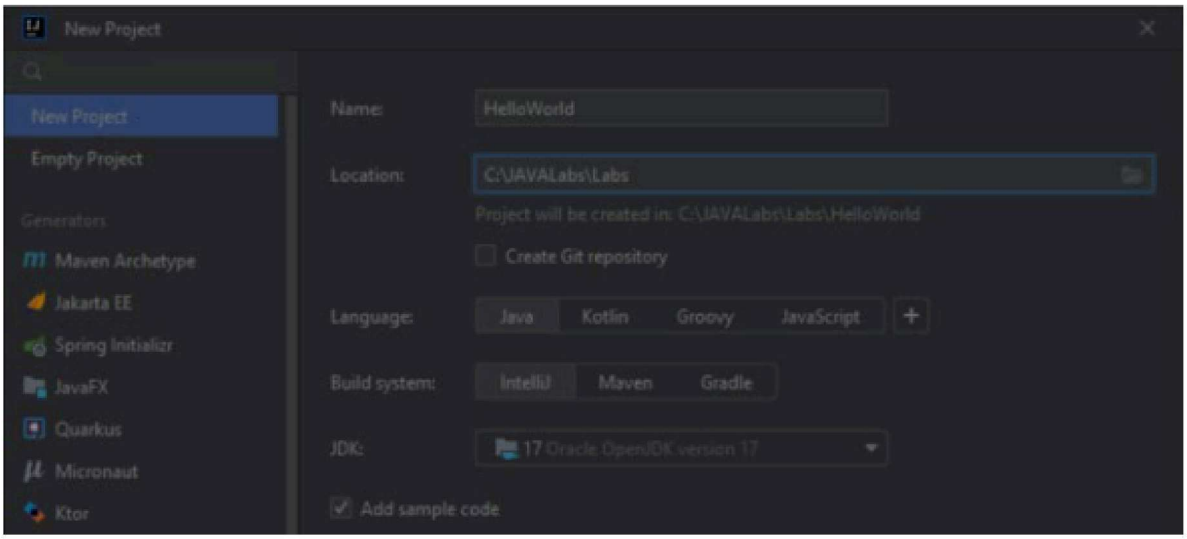
Task 2 – Create an initial Java Project

1. For this initial Project, click New Project on the Welcome screen:

2. On the New Project screen, enter the following details:

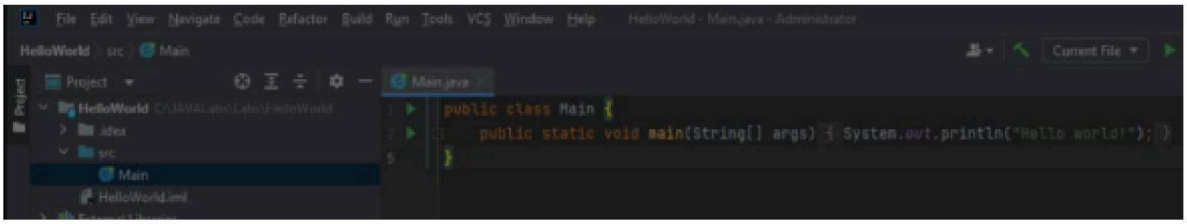
name:	HelloWorld
location:	C:\JAVALabs\Labs
Language:	Java
Build system:	IntelliJ
JDK:	17 Oracle OpenJDK version 17

3. Check the Add sample code box.
Your screen should look like:



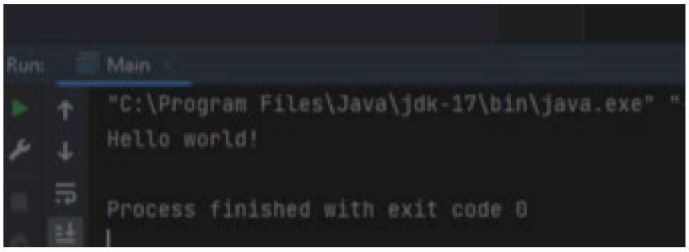
4. Click Create:

The Project window opens with the Project Navigator on the left and the class code editor on the right. The generated class is named Main.

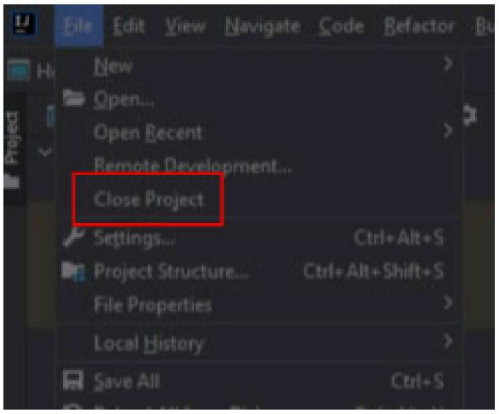


5. Click on the Run button (green triangle) to the top right above the editor screen:

The Run window appears at the bottom of the screen with the application output:



6. Select Close Project from the File menu.



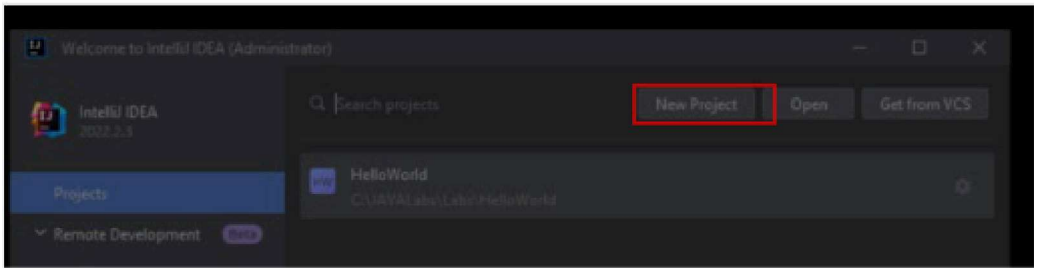
Lab 1 HelloWorld again

The objective of this exercise is to write a new hello world application and run it using both IntelliJ and the command line. We will create a new project and add code ourselves.

Exercise 1-1 – Create a new IntelliJ Project

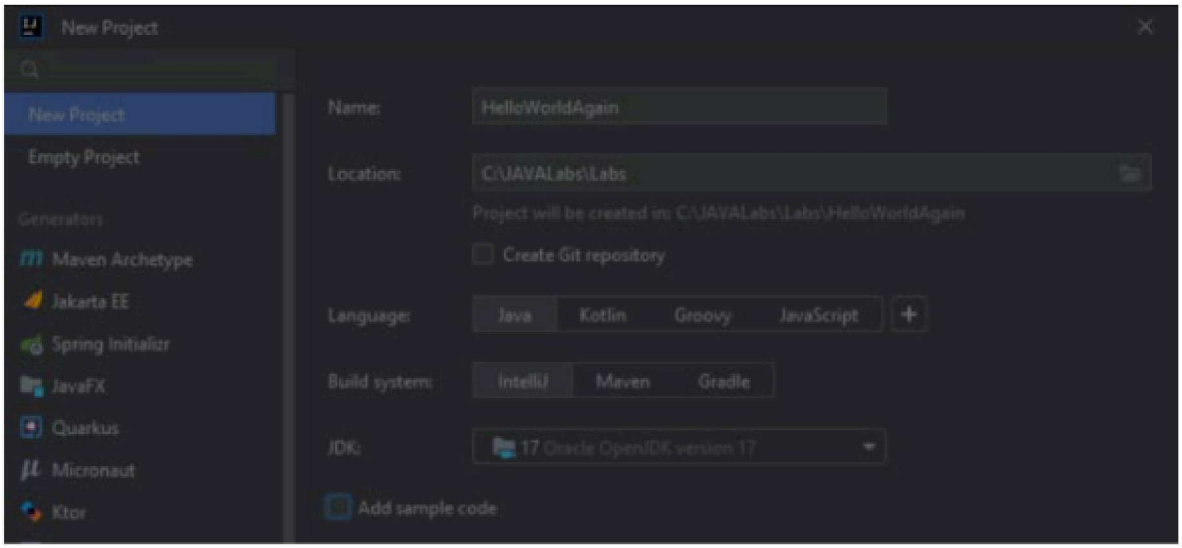
Task 1 – Create a new Project

1. Click the New Project button:



2. On the New Project screen, enter the following details:
   name: HelloWorldAgain
   location: C:\JAVALabs\Labs
   Language: Java
   Build system: IntelliJ
   JDK: 17 Oracle OpenJDK version 17

3. Uncheck the Add sample code box.
   Your screen should look like:

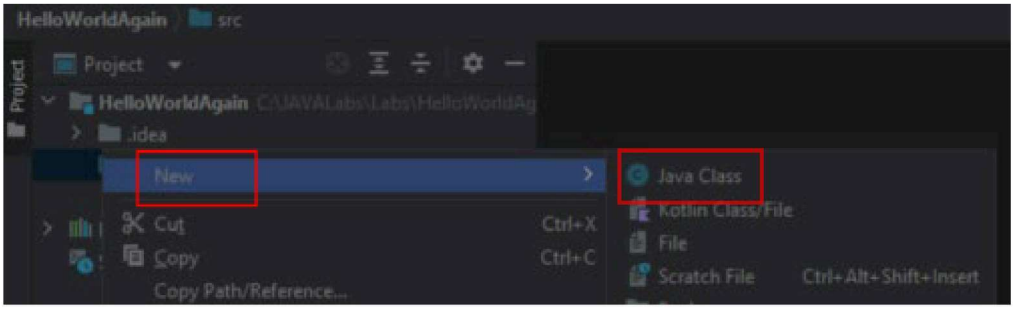


4. Click Create:

The Project window opens with the Project Navigator on the left, but no code editor on the right as no code has been generated this time.

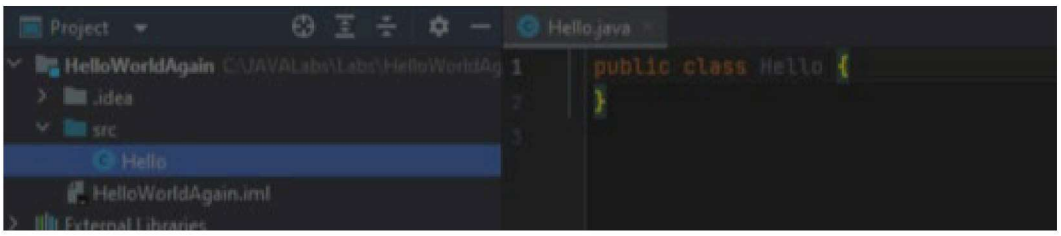
Task 2 – Create a new Java class

1. Right click on the src folder under HelloWorldAgain in the Navigator and select New then Java Class from the context menus.



2. Enter Hello for the New Java Class name and leave Class selected:

Press Enter to create the class source file and open the editor window:

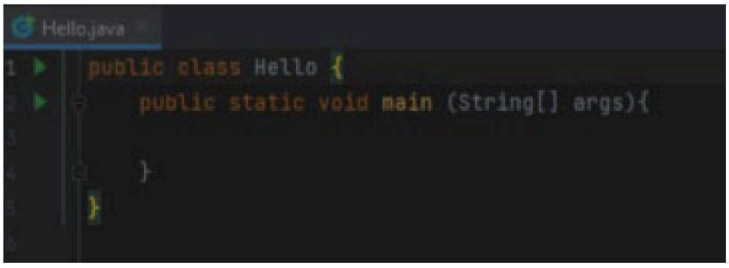


3. Inside the { } braces of the Hello class body code a main method:

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

}
```

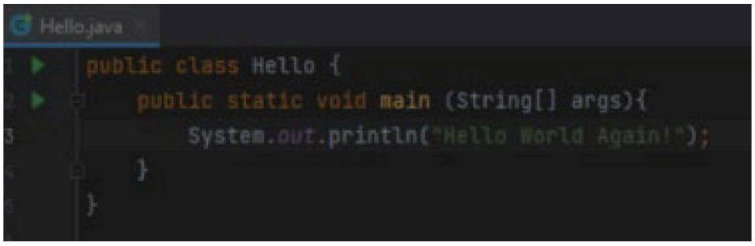
Your code should look like:



4. Inside the { } of the main method code a call to System.out.println(). As you type notice how IntelliJ prompts you with your possible entries:

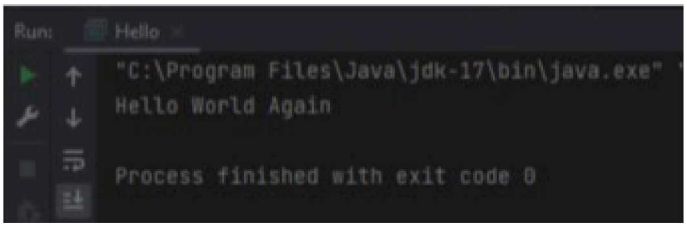
```
System.out.println("Hello World Again!");
```

Your code should look like:



5. Click on the Run button (green triangle) to the top right above the editor screen:

The Run window appears at the bottom of the screen with the application output:

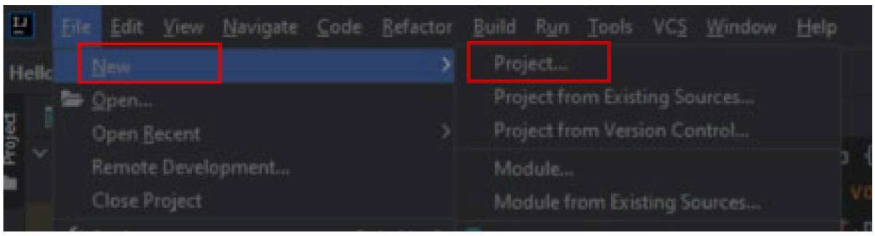


6. Leave this project open for now.

Exercise 1-2 – Create from the File Menu

Task 1 – Create a GreetFriend Project

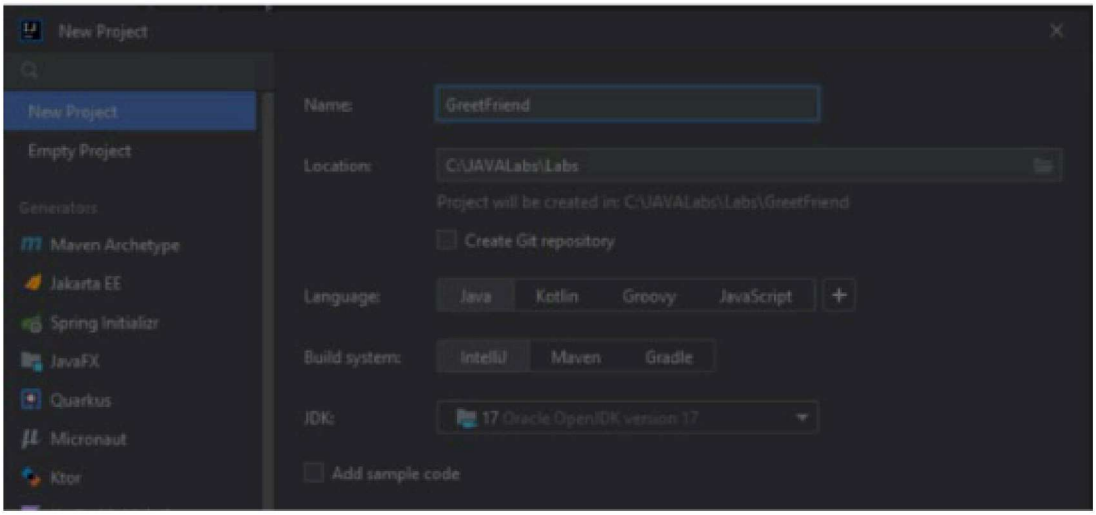
1. Select New then Project from the File menu:



2. On the New Project screen, enter the following details:

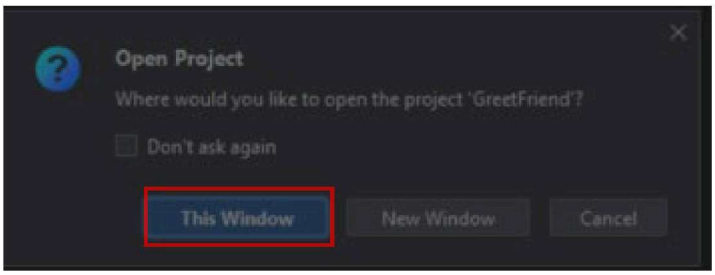
name:	GreetFriend
location:	C:\JAVALabs\Labs
Language:	Java
Build system:	IntelliJ
JDK:	17 Oracle OpenJDK version 17

3. Uncheck the Add sample code box.
Your screen should look like:



4. Click Create:

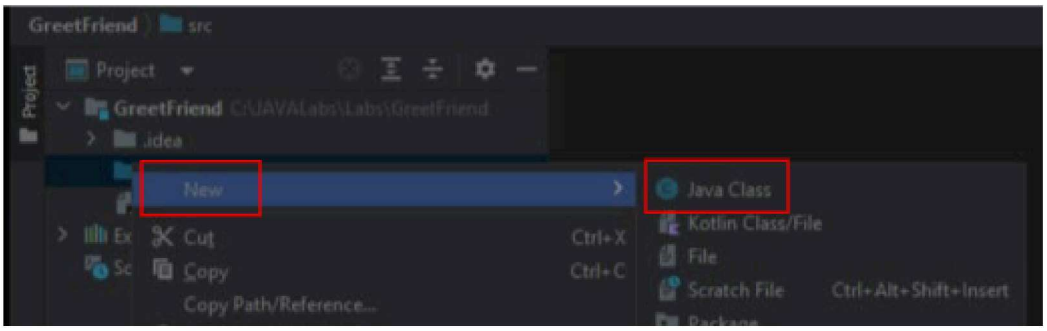
5. As we already have an open project, IntelliJ asks if we want to open the new project in this window or a new window. Select This Window:



The new project replaces the previous one in the Project window. Again, no code has been generated.

Task 2 – Create a new Java class

1. Right click on the src folder under GreetFriend in the Navigator and select New then Java Class from the context menus.



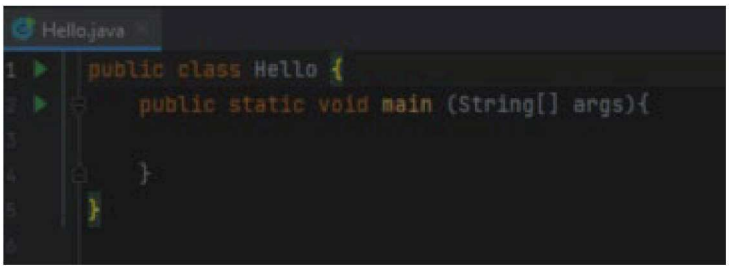
2. Enter Greet for the New Java Class name and leave Class selected:

Click Enter to create the class source file and open the editor window:

3. Inside the { } braces of the Greet class body get IntelliJ to generate a main method. Type psvm immediately followed by the Tab key:

psvm<tab>

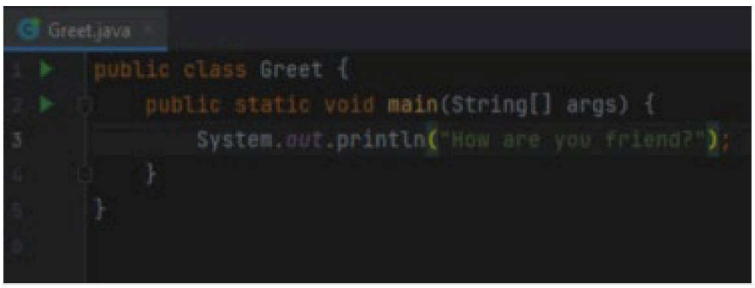
Your code should look like:



4. Inside the { } of the main method body get IntelliJ to generate a call to System.out.println(). Type sout immediately followed by the Tab key:

   sout<tab>

   Add a greeting message. Your code should look like:



5. Click on the Run button (green triangle) to the top right above the editor screen:

   The Run window appears at the bottom of the screen with the application output:

6. Add a String variable called friendName to your main method, initialise it with a name and alter your message to include friendName:

Run your code. Well done on completing your initial lab exercises in IntelliJ. In future lab exercises we will assume that you are familiar with creating Projects, Classes and main() methods!

Exercise 1-3 – Run from the command line

Task 1 – Run Hello from the command line

1. Use your existing terminal window if it's still open, or press the Windows Start button and type:

   cmd

   followed by Enter to open a fresh terminal window.

2. Navigate to the HelloWorldAgain folder where IntelliJ has generated Hello.class and confirm that Hello.class is in the folder:

   ```
   cd C:\JAVALabs\Labs
   cd HelloWorldAgain\out\production\HelloWorldAgain
   dir
   ```

   This will output:

   ```
   C:\Users\Admin>cd C:\JAVALabs\Labs

   C:\JAVALabs\Labs>cd HelloWorldAgain\out\production\HelloWorldAgain

   C:\JAVALabs\Labs\HelloWorldAgain\out\production\HelloWorldAgain>dir
    Volume in drive C has no label.
    Volume Serial Number is 7058-301E

    Directory of C:\JAVALabs\Labs\HelloWorldAgain\out\production\HelloWorldAgain

   15/11/2022  13:57    <DIR>          .
   15/11/2022  13:57    <DIR>          ..
   15/11/2022  13:57               524 Hello.class
                  1 File(s)            524 bytes
                  2 Dir(s)  195,882,967,040 bytes free
   ```

3. Run Hello:

   java Hello

   You should see your output on the screen:

   ```
   C:\JAVALabs\Labs\HelloWorldAgain\out\production\HelloWorldAgain>java Hello
   Hello World Again
   ```

4. Now navigate to your GreetFriend output folder and run your Greet class.

   ```
   C:\JAVALabs\Labs\GreetFriend\out\production\GreetFriend>java Greet
   How are you Prithi?
   ```

Lab 2 Primitives and simple maths

The objective of this exercise is to practice declaring and using basic primitives in code in your IntelliJ project.

Exercise 2-1 – Create primitives

Task 1 – Create a new Project and Class

1. Create a new java project in IntelliJ called Ex02_Primitives.
2. Create a new class called TestVariables and generate a main() method in it. All of the following code will go into this main() method.

Task 2 – Create, initialise and display variables

1. In the main() method create two variables called name and age. Decide on the most appropriate types to use for these.
2. Initialise name and age to suitable values.
3. Print out the values of your name and age variables using System.out.println().
4. Run your code.
5. Modify your print statement to print out a more meaningful phrase, such as "Di's age is 21", using the String concatenation symbol, +, with some text.
6. Assign different values to the name and age and print those out.
7. Run your code again.

Task 3 – Apply some arithmetic

1. Create variables m, x and c to hold values 1, 0.5 and 15 respectively.
2. Create a variable y to hold the fractional result of an equation.
3. Calculate the equation as:
   y = m * x + c
4. Print out each variable value, including the result.
5. Test your code.

Task 4 – Work with booleans

1. Create a boolean variable, isRaining, and give it the value true.
2. Create another boolean, happy, and give it the value true.
3. Combine the two booleans using the logical operators && (and), || (or) and ! (not) to print out the value when:

a. It is raining and we're happy
b. It is raining and we're not happy
c. It is not raining and we're happy
d. It is not raining and we're not happy

4. Test your code.
5. Change the 'and' in the expressions to 'or' and repeat the test. Notice the difference it makes to the output when you use or instead of and.

Solution 2 Primitives and simple maths

The objective of this exercise is to practice declaring and using basic primitives in code in your IntelliJ project.

Exercise 2-1 – Create primitives

Task 1 – Create a new Project and Class

1. Create a new java project in IntelliJ called Ex02_Primitives.
2. Create a new class called TestVariables and generate a main() method in it. All of the following code will go into this main() method.

Task 2 – Create, initialise and display variables

1. In the main() method, create two variables called name and age. Decide on the most appropriate types to use for these.

```
String name;
int age;
```

2. Initialise name and age to suitable values.

```
name = "Alice";
age = 21;
```

3. Print out the values of your name and age variables using System.out.println().

```
System.out.println(name);
System.out.println(age);
```

4. Run your code.
5. Modify your print statement to print out a more meaningful phrase, such as "Di's age is 21", using the String concatenation symbol, +, with some text.

```
System.out.println(name + "'s age is " + age);
```

6. Assign different values to the name and age and print those out.

```
name = "Barak";
age = 100;
System.out.println(name + "'s age is " + age);
```

7. Run your code again.

```
Alice
21
Alice's age is 21
Barak's age is 100
```

Task 3 – Apply some arithmetic

1. Create variables m, x, and c to hold values 1, 0.5, and 15 respectively.

```
int m = 1;
double x = 0.5;
int c = 15;
```

2. Create a variable y to hold the fractional result of an equation.

```
double y;
```

3. Calculate the equation as:
   y = m * x + c

```
y = m * x + c;
```

4. Print out each variable value, including the result.

```
System.out.println("Equation Result");
System.out.println(m + " * " + x + " + " + c + " = " + y);
```

5. Test your code.

```
Equation Result
1 * 0.5 + 15 = 15.5
```

Task 4 – Work with booleans

1. Create a boolean variable, isRaining, and give it the value true.

```
boolean isRaining = true;
```

2. Create another boolean, happy, and give it the value true.

```
boolean happy = true;
```

3. Combine the two booleans using the logical operators && (and), || (or), and ! (not) to print out the value when:
   a. It is raining and we're happy
   b. It is raining and we're not happy
   c. It is not raining and we're happy
   d. It is not raining and we're not happy

```
System.out.println(isRaining && happy);
System.out.println(isRaining && !happy);
System.out.println(!isRaining && happy);
System.out.println(!isRaining && !happy);
```

4. Test your code.

```
true
false
false
false
```

5. Change the 'and' in the expressions to 'or' and repeat the test.

```
System.out.println(isRaining || happy);
System.out.println(isRaining || !happy);
System.out.println(!isRaining || happy);
System.out.println(!isRaining || !happy);
```

Notice the difference it makes to the output when you use or instead of and.

```
true
true
true
false
```

Lab 3 Control flow

The objective of this exercise is to use equality and boolean expressions to write some more complicated code using if, switch and looping.

Some reminders:

Control Flow describes the ways the program runs.

Ternary operator is another form of **if** statement

```
(condition)? value_when_true: value_when_false
```

Switch statements can have multiple possible execution paths

```
switch (expr) {case constant_expr: actions; break;
               case constant_expr: actions; break; …
              default: actions_when_no_case_matched;}
```

Looping means executing repeatedly until a condition is reached

```
while (condition) {repeated_actions;}
do {repeated_actions; } while (condition);
for (initialise; condition; increment ){repeated_actions;}
```

Exercise 3-1 – Looping

Task 1 – Create a new Project and Class

1. Create a new java project in IntelliJ called Ex03_ControlFlow.
2. Create a new class called TestControl and generate a main() method in it. All of the following code will go into this main() method.

Task 2 – while loops

1. Use a while loop to print out pairs of numbers that show the current value of a number, n, and the value of n + 10.
   Start with a value of 0 in n and repeat the process if n is less than 10.
2. Write another while loop that prints out pairs of numbers that show a number, n (with an initial value of 1), and 2^n while n is less than 10.
   a. Create and initialise your result variable before the loop.
   b. In your loop multiply your previous result by 2 to get the next power of 2.

Task 3 – for loops

1. Rewrite these two loops using the for loop construct. Which do you prefer?

Exercise 3-2 – Conditionals

Task 1 – if

1. Write an if statement that, based on a number between 1 and 7, prints out if today is a weekday or a weekend.
    a. Initialise your dayNumber variable to a value between 1 (for Monday) and 7 (for Friday).
    b. Use a chained if to determine whether the dayNumber represents a weekday, a weekend day or an invalid value and print out a message to indicate your finding.
    c. Repeat your test with different dayNumber values.

Task 2 – switch

1. Rewrite the above scenario using a switch statement.
2. If you don't use the break keyword in a switch statement, the execution continues onto the actions of the next case regardless of whether it applied. You may be able to use this to simplify your switch statement.
3. Write a loop which takes the dayNumber from 1 to 7 and prints out whether each day is a weekday or the weekend.

Exercise 3-3 – Leap years (if you have time)

Task 1 – Determining Leap Years

1. Code a loop that prints out all year numbers from 1900 until the current year.
2. Test your code.
3. Add conditions using the modulus operator (%) to work out if the year was a leap year. It is a leap year if:
    a. It is evenly divisible by 4 AND
    b. It is not evenly divisible by 100 OR it is evenly divisible by 400
4. Print " was a leap year" or " was not a leap year" as appropriate with each year. For example, 1900 was not a leap year, 1904 was a leap year, 2000 was a leap year, and 2022 was not a leap year.
5. Amend your code to only print the leap years.

Solution 3 Control Flow

```
public class TestControl {
	public static void main(String[] args) {
		// Ex3-1 Task2-1
		int x = 0;
		while (x < 10) {
			// this can be done in one line
			// or a separate variable can be used to calculate
			// x + 10
			System.out.println("(" + x + ", " + (x + 10) + ")");
			x++;
		}
		System.out.println("***************");
		// Ex3-1 Task2-2
		int n = 1;
		int sum = 1;
		while (n < 10) {
			sum = sum * 2;
			System.out.println("(" + n + ", " + sum + ")");
			n++;
		}
		System.out.println("***************");
		// Ex3-1 Task3
		for (int i = 0; i < 10; i++) {
			System.out.println("(" + i + ", " + (i + 10) + ")");
		}
		System.out.println("***************");
		//
		int sum2 = 1;
		for (int i = 0; i < 10; i++) {
			sum2 = sum2 * 2;
			System.out.println("(" + i + ", " + sum2 + ")");
		}
		System.out.println("***************");
		// Ex3-2 Task1
		int dayOfWeek = 2;
		// assuming the week starts on monday!
		if (dayOfWeek == 1) {
			System.out.println("It's a weekday");
		} else if (dayOfWeek == 2) {
			System.out.println("It's a weekday");
		} else if (dayOfWeek == 3) {
```

```
        System.out.println("It's a weekday");
} else if (dayOfWeek == 4) {
        System.out.println("It's a weekday");
} else if (dayOfWeek == 5) {
        System.out.println("It's a weekday");
} else if (dayOfWeek == 6) {
        System.out.println("It's a weekend");
} else if (dayOfWeek == 7) {
        System.out.println("It's a weekend ");
} else {
        System.out.println(
        "Not sure where you get days from. Use 1 to 7");
}
// an alternative using if
if (dayOfWeek > 0 && dayOfWeek < 6) {
        System.out.println("It's a weekday");
} else if (dayOfWeek == 6 || dayOfWeek == 7) {
        System.out.println("It's a weekend");
} else {
        System.out.println(
        "Not sure where you get days from. Use 1 to 7");
}
// Ex3-2 Task2-1

switch (dayOfWeek) {
case 1:
        System.out.println("It's a weekday");
        break;
case 2:
        System.out.println("It's a weekday");
        break;
case 3:
        System.out.println("It's a weekday");
        break;
case 4:
        System.out.println("It's a weekday");
        break;
case 5:
        System.out.println("It's a weekday");
        break;
case 6:
        System.out.println("It's a weekend");
```

```
        break;
case 7:
        System.out.println("It's a weekend");
        break;
default:
        System.out.println(
        "Not sure where you get days from. Use 1 to 7");
}
// Ex3-2 Task2-2
switch (dayOfWeek) {
case 1:
case 2:
case 3:
case 4:
case 5:
        System.out.println("It's a weekday");
        break;
case 6:
case 7:
        System.out.println("It's a weekend");
        break;
default:
        System.out.println(
        "Not sure where you get days from. Use 1 to 7");
}
System.out.println("************************");
// Ex3-2 Task2-3
for (int i = 1; i < 8; i ++){
        switch (i) {
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
                System.out.println("It's a weekday");
                break;
        case 6:
        case 7:
                System.out.println("It's a weekend");
                break;
        default:
                System.out.println("Invalid day. Use 1 to 7");
```

```
            }
        }
        System.out.println("************************");

        // Ex3-3 Task1
        for(int year = 1900; year < 2023; year++){
            if ((year % 4 == 0) &&  (year % 100 != 0) ||
                                    (year % 400 == 0)) {
                System.out.println(year + " was a leap year");
            }
        }
    }
}
```

Lab 4 Introduction to Objects

The objective of this exercise is to understand how to create and use objects in Java.

This exercise is in two parts. First, we're going to create some objects based around a scenario and then you're going to model something of your own.

With all these exercises, it is best to test each step as you write. To do this with objects create a test class with a main method in it, then create the objects in this method and print out the various fields and output from methods as you write them. For example, after writing a setName() method, a good test is to create an object of that type, call setName(), then call getName(), printing out the results.

Key Terms

Object

is the combination of state and behaviour.

Programming object

is an instance of a class.

Class

is a specification of a real-world object and acts as a placeholder for variables and methods.

Constructor method

is a special member method which is called when the Java class is instantiated.

Method

is a block of code which can be declared once and called many times.

Instantiation

is the process of creating objects from the class.

Encapsulation

Is the process of combining variables and their methods into a separate logical unit.

Static members

are members (variables and methods) that can be called without an instance of the object being created.

Boilerplate code

refers to simple methods, classes, and annotation that you will end up writing repeatedly.

Exercise 4-1 – books

Task 1 – Create a new project and class

1. Create a new java project in IntelliJ called Ex04_Objects.
2. Create a package called com.qa.
3. Create a new class called com.qa.TestBooks and generate a main() method in it. Your code to create and work with Book objects will go into this main() method.

Task 2 – Create a new Book Class

1. Create a class that describes a Book in the com.qa package. It should have a name, author, and price.
2. Write a constructor that sets all the fields in the class.
3. Create getter and setter methods for each of the fields in the class. Make sure that you set your fields to private, as this is both good practice and good for security.
4. Create a toString() method by to return all fields concatenated together.
5. In your test class, create a number of books and print out their contents.
6. Declare an array variable to store up to three Book objects.
7. Add the books (Book objects) to the array created in the previous step.
8. Loop through the array printing out the contents of each object

Exercise 4-2 – Create your own Objects

Task 1 – Create a Class for your own Object type

1. Think of a real world object and model it in Java form. Think about the fields you will need to describe the object and the methods that will act on it.
2. Then write the class for your object type in the com.qa package including a constructor, accessor and mutator methods, and a toString() method.
3. Have a play around with scope and the differences between public and private.
4. Use the main() method in TestBooks to create some of your objects.

Solution 4 Introduction to Objects

```
package com.qa;
import java.util.Arrays;
public class Book {
	private String name;
	private String[] authors;
	private double price;
	// Ex4-1 Task2-2
	public Book(String name, String[] authors, double price) {
		this.name = name;
		this.authors = authors;
		this.price = price;
	}
	// Ex4-1 Task2-3
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String[] getAuthors() {
		return authors;
	}
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
	// Ex4-1 Task2-4
	@Override
	public String toString() {
		return "Book [name=" + name + ", authors=" +
		   Arrays.toString(authors) + ", price=" + price + "]";
	}
}
```

```
package com.qa;
public class Main {
	public static void main(String[] args) {
		String[] arr = {"Author1", "Author2", "Author3"};
		// Ex4-1 Task2-5
```

```
        Book b1 = new Book("title", arr, 10.99);
        Book b2 = new Book("title2", arr, 5.99);
        Book b3 = new Book("title3", arr, 1.00);
        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        // Ex4-1 Task2-6
        Book[] bookArray = new Book[3];
        // Ex4-1 Task2-7
        bookArray[0] = b1;
        bookArray[1] = b2;
        bookArray[2] = b3;
        //this could have been done with
        //          "Book[] bookArray = {b, b2, b3};"
        // Ex4-1 Task2-8
        for (int x = 0; x < bookArray.length; x++){
            System.out.println(bookArray[x]);
        }
    }
}
```

Lab 5 Inheritance and Polymorphism

The objective of this exercise is to look at object hierarchies and implement some classes which extend from one another and use interfaces to define behaviours.

In this exercise, we are going to work with three classes, Shape, Rectangle, and Circle. There is also a 'helper' class called Point which stores an x and y coordinate pair allowing us to return the current coordinates of the shape.

Key terms

Inheritance

is the process of creating a new class from an existing class.

Abstract class

allows for the parent in a relationship to describe what methods should be in the subclasses without providing an implementation.

Method Overriding

is when a subclass overrides, or provides its own version of, its super class methods without modifying the method prototype.

Exercise 5-1 – Inheritance

Task 1 – Create a new Project and Package

1. Create a new java project in IntelliJ called Ex05_Inheritance.
2. Create a package called com.qa.

Task 2 – Create the Point class

1. Create the Point class in com.qa.
2. Assign it two private double fields, x and y.
3. Generate a constructor, getters, setters, and a toString() method for the class.

Task 3 – Create the Shape class

1. Create the Shape class in com.qa.
2. Shape has four fields, a String name, a String colour, and two doubles for the position called x and y.
3. Write a constructor, getters, setters, and a toString() method for the class.
4. Make the class abstract and add two abstract methods:

a. getArea() that returns a double

b. getCentrePoint() that returns a Point object.

Task 4 – Create the Rectangle class

1. Create a class called Rectangle in com.qa that extends Shape.
2. A Rectangle is a type of Shape that needs to store the height and width of the rectangle as doubles.
3. Write the constructor for the Rectangle class.
    a. It should take in all the parameters required to setup the Shape object and the height and width parameters required for the Rectangle.
    b. Remember to use super(...) to call Shape's constructor with the required fields.
4. It's always useful to have a toString() method for testing the class, so either write one yourself or automatically generate one using IntelliJ's built in option.
5. Implement a method isSquare() which returns a boolean value stating true if the rectangle is a square (if the height and the width are the same), or false otherwise.
6. Finally, we need to override the abstract methods in the Shape class. Implement the getArea() method and the getCentrePoint() method for the rectangle class.
    a. The area is calculated as the width multiplied by the height.
    b. The centre point coordinates are calculated as x plus half the width and y plus half the height. Use the getX() and getY() methods inherited from Shape to access the x and y values.

Task 5 – Create the Main class

1. Test your class by creating a Main class in com.qa with a main method in it.
2. In this, create a few Rectangle objects and print out the contents of the objects using the toString() method.
3. Call the getArea() and getCentrePoint() methods on your Rectangle objects and print the results.

Task 6 – Create the Circle class

1. Create a class called Circle in com.qa that extends Shape.
2. A Circle is a type of Shape that needs to store the radius of the Circle as a double.
3. Write the constructor for the Circle class.

   a. It should take in all the parameters required to setup the Shape object and the radius parameter required for the Circle.

   b. Use super(...) to call Shape's constructor with the required fields.

4. Implement a toString() method for testing the class, either by writing your own or getting IntelliJ to generate one.

5. Override the abstract methods in the Shape class. Implement the getArea() method and the getCentrePoint() method for the Circle class.

   a. The area is calculated as πr^2. The static constant Math.PI is available in the java.lang package.

   b. The centre point coordinates are those held in the x and y fields. Use the getX() and getY() methods inherited from Shape to access the x and y values required to construct the Point return object.

6. Create some Circle objects in your main() method and print out the contents of the objects using the toString() method.

7. Call the getArea() and getCentrePoint() methods on your Circle objects and print the results.

Task 7 – Enhance the toString() methods (if you have time)

1. Enhance your code by including calls to getArea(), getCentrePoint() and isSquare() in the toString() implementations

Solution 5 Inheritance and Polymorphism

```
package com.qa;
public class Point {
	private double x, y;
	public Point(double x, double y){
		this.x = x; this.y = y;
	}
	public double getX() {
		return x;
	}
	public void setX(double x) {
		this.x = x;
	}
	public double getY() {
		return y;
	}
	public void setY(double y) {
		this.y = y;
	}
	@Override
	public String toString() {
		return "Point [x=" + x + ", y=" + y + "]";
	}
}
package com.qa;
// abstract means that this class can't be instantiated, but can
// have abstract methods as well as concrete methods
public abstract class Shape {
	//private fields
	private String name;
	private String colour;
	private double x, y;
	//basic constructor
	public Shape(String name, String colour, double x, double y) {
		this.name = name;
		this.colour = colour;
		this.x = x;
		this.y = y;
	}
	//getters and setters
	public double getX() {
		return x;
```

```
	}
	public void setX(double x) {
		this.x = x;
	}
	public double getY() {
		return y;
	}
	public void setY(double y) {
		this.y = y;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getColour() {
		return colour;
	}
	public void setColour(String colour) {
		this.colour = colour;
	}
	//abstract methods - implemented by the sub-classes
	public abstract double getArea();
	public abstract Point getCentrePoint();
}
package com.qa;
public class Rectangle extends Shape{
	//fields, not available in Shape
	private double height, width;
	//constructor that calls the Shape constructor
	public Rectangle(String name, String colour, double x,
			double y, double width, double height) {
		super(name, colour, x, y);
		this.width = width;
		this.height = height;
	}
	//method - not available in Circle
	public boolean isSquare(){
		return (width == height);
	}
	//override the abstract methods in the Shape class
```

```
	@Override
	public double getArea() {
		return width * height;
	}
	@Override
	public Point getCentrePoint() {
		double centreX = getX() + width/2;
		double centreY = getY() + height/2;
		return new Point(centreX, centreY);
	}
	//override the method in the Object class
	@Override
	public String toString() {
		return "Rectangle [height=" + height + ", width=" + width
		  + ", isSquare()=" + isSquare() + ", getArea()=" +
		 getArea() + ", getCentrePoint()=" + getCentrePoint()
		  + ", getX()=" + getX() + ", getY()=" + getY() +
		 ", getName()=" + getName()+ ", getColour()=" +
		 getColour() + "]";
	}
}
package com.qa;
public class Circle extends Shape{
	private double radius;
	public Circle(String name, String colour, double x, double y,
			double radius) {
		super(name, colour, x, y);
		this.radius = radius;
	}
	@Override
	public double getArea() {
		return Math.PI * radius * radius;
	}
	@Override
	public Point getCentrePoint() {
		return new Point(getX(), getY());
	}
	@Override
	public String toString() {
		return "Circle [radius=" + radius + ", getArea()=" +
			getArea() + ", getCentrePoint()=" + getCentrePoint()
			+ ", getX()=" + getX() + ", getY()=" + getY() +
```

```
                ", getName()=" + getName()  + ", getColour()=" +
                getColour() + "]";
      }
}
package com.qa;
public class Main {
      public static void main(String[] args) {
            //create some objects
            Circle c = new Circle ("circle1", "red", 0,0, 10);
            Circle c2 = new Circle ("circle2", "blue", 10,10, 5);
            Rectangle r = new Rectangle ("rectangle1", "yellow",
                                         0, 0, 10, 10);
            Rectangle r2 = new Rectangle ("rectangle2", "green",
                                         2, 2, 5, 2);
            //print out the objects
            System.out.println(c);
            System.out.println(c2);
            System.out.println(r);
            System.out.println(r2);
/* following code not required when toString() methods enhanced
            System.out.println(c + ", CentrePoint is: " +
                  c.getCentrePoint() + ", Area is: " + c.getArea());
            System.out.println(c2 + ", CentrePoint is: " +
                  c2.getCentrePoint() + ", Area is: " + c2.getArea());
            System.out.println(r + ", CentrePoint is: " +
                  r.getCentrePoint() + ", Area is: " + r.getArea()
                  + ". Is it square? " + r.isSquare());
            System.out.println(r2 + ", CentrePoint is: " +
                  r2.getCentrePoint() + ", Area is: " + r2.getArea()
                  + ". Is it square? " + r2.isSquare());
*/
      }
}
```

Lab 6 Interfaces

The objective of this lab is to look at object hierarchies and implement some classes which extend from one another and use interfaces to define behaviours.

This lab extends Lab 5 (Inheritance) and creates an interface which allows an implementation of the Shape to be Movable. We want the Circle class to be Movable. As we can't inherit from more than one class, we will use an interface to provide this functionality. Interfaces can be thought of as "jobs the object can also do" and inheritance as "what type the object is".

The last part of the lab demonstrates the use of default methods in interfaces.

Key terms:

Interface

an interface is a specification of method signatures.

Method Overriding

is when a subclass overrides, or provides its own version of, its super class methods without modifying the method prototype.

Exercise 6-1 – Working with an interface

Task 1 – Create an interface

1. Create a new project Ex06_Interfaces and copy the com.qa package from Ex5_Objects. Create a new interface called com.qa.Movable. It's declaration should be:
```
public interface Movable {
}
```
2. Declare two abstract methods in the interface:
    a. getCurrentLocation() takes no parameters and returns a Point object.
    b. move() takes in two double parameters, x and y, and returns nothing.

Task 2 – Implement an interface

1. Amend the Circle class declaration so that it implements Movable.
2. We now need to implement the abstract methods to make Circle valid again:
    a. getCurrentLocation() takes no parameters and returns a Point object. It should just return the current location of the object, which

is the same as the centre point for the Circle. The getX() and getY() methods from the superclass will provide the arguments for the Point() constructor.

b. move() takes in two double parameters, x and y, and returns nothing. It should change the current x and y coordinates of the object by adding to them the values passed in as the parameters. Use the setX() and setY() superclass methods to achieve this.

3. Finally, to test the methods:

   a. Create a Circle object in the main class.

   b. Print the Point object returned by the Circle's getCurrentLocation() method to show the current position.

   c. Call the move() method to change its location.

   d. Print out the contents of the object or the result from getCurrentLocation(). Check that the x and y coordinates have changed. If not, you might want to use the debugger to see what is going on as the program runs.

Solution 6-1 Working with an interface

```
package com.qa;
public interface Movable {
	public Point getCurrentLocation();
	public void move(double x, double y);
}
```

```
package com.qa;
public class Circle extends Shape implements Movable{
	double radius;
	public Circle(String name, String colour, double x, double y,
			double radius) {
		super(name, colour, x, y);
		this.radius = radius;
	}
	@Override
	public double getArea() {
		return Math.PI * radius * radius;
	}
	@Override
	public Point getCentrePoint() {
		return new Point(super.getX(), super.getY());
	}
	@Override
	public String toString() {
		return "Circle [radius=" + radius + ", getArea()=" +
		getArea() + ", getCentrePoint()=" + getCentrePoint() +
		", getX()=" + getX() + ", getY()=" + getY() +
		", getName()=" + getName() + ", getColour()=" +
		getColour() + "]";
	}
	public Point getCurrentLocation() {
		return getCentrePoint();
```

```
	}
	public void move(double x, double y) {
		setX(getX() + x);
		setY(getY() + y);
	}
}
```

```
package com.qa;
public class Main {
	public static void main(String[] args) {
		//create some objects
		Circle c = new Circle ("circle1", "red", 0,0, 10);
		Circle c2 = new Circle ("circle2", "blue", 10,10, 5);
		Rectangle r = new Rectangle ("rectangle1", "yellow",
					0, 0, 10, 10);
		Rectangle r2 = new Rectangle ("rectangle2", "green",
					2, 2, 5, 2);
		//print out the objects
		System.out.println(c);
		System.out.println(c2);
		System.out.println(r);
		System.out.println(r2);

		System.out.println("Circle location before move: " +
					c.getCurrentLocation());
		c.move(10.5, 4.3);
		System.out.println("Circle location after move: " +
					c.getCurrentLocation());
	}
}
```

Exercise 6-2 – Virtual extension methods

Task 1 – Create Interfaces with default Methods

1. In the Ex06_Interfaces project create a new public interface called com.qa.Loggable.
    a. It should have one abstract method, log(), which takes a String message and returns void.
2. Create a ConsoleLoggable interface which extends Loggable and provides a default implementation of the log() method and writes to the output window with a "Console Output> " prefix.
3. Create a TimeLoggable interface which extends Logger and provides a different default implementation of the log() method, printing out the current time before the message, using CurrentDateTime.now().
4. Create a new basic class Triangle that implements ConsoleLoggable.
5. Create a new basic class Pentagon that implements TimeLoggable.
6. Create a new basic class Hexagon that implements both ConsoleLoggable and TimeLoggable.
    a. Notice that Java isn't happy that log() is inherited from both interfaces.
    b. Force your class to use the TimeLoggable implementation of the log() method.
7. In your main() method create objects of types Triangle, Pentagon, and Hexagon and call their log() methods with suitable messages.

Solution 6-2 Virtual extension methods

```
package com.qa;

public interface Loggable {
    public void log(String message);
}
```

```
package com.qa;

public interface ConsoleLoggable extends Loggable{
    public default void log(String msg) {
        System.out.println("Console> " + msg);
    }
}
```

```
package com.qa;

import java.time.LocalDateTime;

public interface TimeLoggable extends Loggable{
    public default void log(String msg) {
        System.out.println(LocalDateTime.now() + "> " + msg);
    }
}
```

```
package com.qa;

public class Triangle extends Shape implements ConsoleLoggable{
    private double side1, side2, side3;

    public Triangle(String name, String colour, double x, double y,
                    double side1, double side2, double side3) {
        super(name, colour, x, y);
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    public double getSide1() {
        return side1;
    }

    public void setSide1(double side1) {
        this.side1 = side1;
    }

    public double getSide2() {
        return side2;
    }

    public void setSide2(double side2) {
```

```
        this.side2 = side2;
    }

    @Override
    public String toString() {
        return "Triangle{" +
                "side1=" + side1 +
                ", side2=" + side2 +
                ", side3=" + side3 +
                '}';
    }

    public double getSide3() {
        return side3;
    }

    public void setSide3(double side3) {
        this.side3 = side3;
    }

    @Override
    public double getArea() {
        // Heron's formula
        return 0.25 * Math.sqrt((4 * side1 * side1 * side2 * side2)
               - Math.pow(((side1 * side1) + (side2 * side2)
                           - (side3 * side3)), 2) );
    }

    @Override
    public Point getCentrePoint() {
        // to be developed
        return null;
    }
}
```

```
package com.qa;

public class Pentagon extends Shape implements TimeLoggable{
    private double side1, side2, side3, side4, side5;

    public Pentagon(String name, String colour, double x, double y,
                    double side1, double side2, double side3,
                    double side4, double side5) {
        super(name, colour, x, y);
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
        this.side4 = side4;
        this.side5 = side5;
    }

    public double getSide1() {
        return side1;
    }
```

```
@Override
public String toString() {
    return "Pentagon{" +
            "side1=" + side1 +
            ", side2=" + side2 +
            ", side3=" + side3 +
            ", side4=" + side4 +
            ", side5=" + side5 +
            '}';
}

public void setSide1(double side1) {
    this.side1 = side1;
}

public double getSide2() {
    return side2;
}

public void setSide2(double side2) {
    this.side2 = side2;
}

public double getSide3() {
    return side3;
}

public void setSide3(double side3) {
    this.side3 = side3;
}

public double getSide4() {
    return side4;
}

public void setSide4(double side4) {
    this.side4 = side4;
}

public double getSide5() {
    return side5;
}

public void setSide5(double side5) {
    this.side5 = side5;
}

@Override
public Point getCentrePoint() {
    // to be developed
    return null;
}

@Override
public double getArea() {
    // to be developed
```

```
        return 0;
    }
}
package com.qa;

public class Hexagon extends Shape implements ConsoleLoggable,
TimeLoggable{
    private double side1, side2, side3, side4, side5, side6;

    @Override
    public void log(String msg) {
        TimeLoggable.super.log(msg);
    }


    public Hexagon(String name, String colour, double x, double y,
double side1, double side2, double side3, double side4, double
side5, double side6) {
        super(name, colour, x, y);
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
        this.side4 = side4;
        this.side5 = side5;
        this.side6 = side6;
    }

    public double getSide1() {
        return side1;
    }

    public void setSide1(double side1) {
        this.side1 = side1;
    }

    @Override
    public String toString() {
        return "Hexagon{" +
                "side1=" + side1 +
                ", side2=" + side2 +
                ", side3=" + side3 +
                ", side4=" + side4 +
                ", side5=" + side5 +
                ", side6=" + side6 +
                '}';
    }

    public double getSide2() {
        return side2;
    }

    public void setSide2(double side2) {
        this.side2 = side2;
    }

    public double getSide3() {
```

```
        return side3;
    }

    public void setSide3(double side3) {
        this.side3 = side3;
    }

    public double getSide4() {
        return side4;
    }

    public void setSide4(double side4) {
        this.side4 = side4;
    }

    public double getSide5() {
        return side5;
    }

    public void setSide5(double side5) {
        this.side5 = side5;
    }

    public double getSide6() {
        return side6;
    }

    public void setSide6(double side6) {
        this.side6 = side6;
    }
    @Override
    public double getArea() {
        // to be developed
        return 0;
    }

    @Override
    public Point getCentrePoint() {
        // to be developed
        return null;
    }

 }
```

```
package com.qa;

public class Main {

      public static void main(String[] args) {

            //create some objects

            Circle c = new Circle ("circle1", "red", 0,0, 10);

            Circle c2 = new Circle ("circle2", "blue", 10,10, 5);

            Rectangle r = new Rectangle ("rectangle1", "yellow",
```

```
                                    0, 0, 10, 10);
        Rectangle r2 = new Rectangle ("rectangle2", "green",
                                    2, 2, 5, 2);
        //print out the objects
        System.out.println(c);
        System.out.println(c2);
        System.out.println(r);
        System.out.println(r2);

        System.out.println("Circle location before move: " +
                           c.getCurrentLocation());
        c.move(10.5, 4.3);
        System.out.println("Circle location after move: " +
                           c.getCurrentLocation());
        Triangle t1 = new Triangle("Tripod", "Green", 3.2, 5.8,
                                   1, 2, 3);
        Pentagon p1 = new Pentagon("White House", "White",
                                   432678.2, 9876547.8, 90807,
                                   80605, 65430, 12345, 67890);
        Hexagon h1 = new Hexagon("Pie", "Brown", 0, 0, 1, 2, 3,
                                 4, 5, 6);
        t1.log(t1.getName() + " checking in");
        p1.log(p1.getName() + " signing in");
        h1.log(h1.getName() + " very tasty");

    }
}
```

Lab 7 Collections

This lab looks at how to use collections and generics and we'll write some more simple objects. By the end of this exercise, you should be more comfortable with creating and using collections.

The API Documentation is particularly useful when dealing with the various types of collection. Refer to it as you work with each type to check which methods are available.

Key Terms

Collection

is a group of java objects.

List

is an interface which maintains the order of insertion and allows duplication.

Set

is an interface which organises the collection objects based on their hashcode. It does not allow duplication.

Map

represents the objects based on the key-value pairs.

ArrayList

is one of the implementation classes of the List interface.

HashMap

is an implementation class of the Map interface which maintains the objects based on key-value pair. HashMap uses hashcode.

Generic class

is a class that provides type-safety and avoids explicit type casting.

Exercise 7-1 – Project Setup

Task 1 – Create a new project and package

1. Create a new java project in IntelliJ called Ex07_Collections.
2. Create a package called com.qa.
3. Inside the com.qa package create the class Main with a main method. This method will be where your collections code goes later.

Task 2 – Create a package containing Animal classes

1. Create another package, com.qa.model.
2. Inside the com.qa.model package create four java classes with the following code (feel free to copy from Ex07-CollectionsSkeleton in the Labs folder).

Animal.java

```
package com.qa.model;
public abstract class Animal {
	//private fields
	private	String name;
	private	int	age;
	//constructor
	public Animal(String name, int age){
		this.name = name;
		this.age  = age;
	}
	//setters and getters for name and age
	public void setName(String name) {
		this.name = name;
	}
	public String getName() {
		return name;
	}
	public void setAge(int age) {
		this.age = age;
	}
	public int getAge() {
		return age;
	}
	//Abstract methods, the Animal class doesn't know
	//how to implement these. We need some concrete
	//instantiations in the subclasses
	public abstract String sayHello();
```

```
	public abstract String move();
	public String toString(){
		return "Name: " + name + " Age: " + age;
	}
}
```

Cat.java

```
package com.qa.model;
public class Cat extends Animal{
	public Cat(String name, int age) {
		super(name, age);
	}
	@Override
	public String sayHello() {
		return "Oh. It's you. Hello." + getName();
	}
	@Override
	public String move() {
		return "If you insist...";
	}
	public String toString(){
		return "Cat: " + super.toString();
	}
}
```

Dog.java

```
package com.qa.model;
public class Dog extends Animal {
	public Dog(String name, int age) {
	    super(name, age);
	}
	@Override
	public String sayHello() {
	    return "Oh hello! You're back! Hello! Hello! "
	            + "I missed you! Hello!";
	}
	@Override
	public String move() {
		return "Ruuuuuuuuuun!";
	}
	public String toString(){
		return "Dog: " + super.toString();
```

```
	}
}
```

Rabbit.java

```
package com.qa.model;
public class Rabbit extends Animal {
	public Rabbit(String name, int age) {
		super(name, age);
	}
	@Override
	public String sayHello() {
		return "Snuffle snuffle";
	}
	@Override
	public String move() {
		return "Hop hop hop";
	}
	public String toString(){
		return "Rabbit: " + super.toString();
	}
}
```

Exercise 7-2 – Creating collections

Task 1 – Create Animal objects

1. Create a variety of Animal objects from the classes in com.qa.model. You will need at least 5 to put in different types of collections.

Task 2 – Using an ArrayList

1. Create an ArrayList of all your Animal objects, giving the type using <>.
2. Look at the difference when you give it the subtype (Cat, Dog, Rabbit) rather than the supertype (Animal).
3. Use the debugger to inspect the ArrayList object as the various Animals are being added to it.
4. Print out the values in your ArrayList using a for loop, a for-each loop, or the iterator. Try all three methods.

Task 3 – Using HashMaps

1. Create a HashMap that uses the name of the Animal as a key and the Animal object as the value.
2. Populate it with your objects.
3. Create a second HashMap that uses the object as the key and a description for the value.
4. Populate the second HashMap with your objects and suitable descriptions for each.
5. Print out the values in your HashMaps.

Task 4 – Using a Set

1. Create a HashSet of Animal objects and populate it with your objects.
2. Attempt to add an object to the Set a second time. Will it allow duplicates?
3. Print out the values in your Set.

Exercise 7-3 – Going further with collections (if you have time)

Task 1 – Find a specific object

1. Find a specific object in your collections. Choose one of your objects (for example, a cat called "Bob"). Look through the API Documentation to see if there are any methods that can help you. For some collections you need to iterate through each element, for others you can just go directly to the object in question.

Task 2 – Sort your List

1. Now we want to sort our ArrayList. We have a sort() method to do this for a List in the Collections class, but it needs to know how to sort the objects in the List.
2. First, we need the Animal class to extend the Comparable interface and implement the compareTo() method in either the superclass or each of the sub-classes. Sort the animals by their age.
3. Sort the contents of your ArrayList by using the Collections.sort() method.
4. Display the values in your sorted ArrayList.

Task 3 – Create a TreeMap

1. We can't use the sort() method on a Set or a HashMap. Look at the Java API and see if there are any other types of maps or sets you can use that will sort the collections for us.
2. Implement one of these collections, perhaps a TreeMap or SortedSet, and populate it with your objects.
3. Output the elements from your sorted collection.

Solution 7 Collections

Main.java

```
package com.qa;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.TreeMap;
import com.qa.model.Animal;
import com.qa.model.Cat;
import com.qa.model.Dog;
import com.qa.model.Rabbit;
public class Main {
	public static void main(String[] args) {
		// Ex7-2 Task1
		Cat c = new Cat("Whiskers", 10);
		Cat c2 = new Cat("Midas", 8);
		Dog d = new Dog("Spot", 2);
		Rabbit r = new Rabbit("Peter", 2);
		Rabbit r2 = new Rabbit("Cottontail", 3);
		// Ex7-2 Task2
		ArrayList<Animal> animalList = new ArrayList<Animal>();
		animalList.add(c);
		animalList.add(c2);
		animalList.add(d);
		animalList.add(r);
		animalList.add(r2);
		ArrayList<Cat> catList = new ArrayList<Cat>();
		catList.add(c);
		catList.add(c2);
//		catList.add(d); // error!
		System.out.println("For Loop on ArrayList");
		for (int x = 0; x < animalList.size(); x ++){
			System.out.println(animalList.get(x));
		}
		// Ex7-2 Task3
		HashMap<String, Animal> animalMap =
				new HashMap<String, Animal>();
		animalMap.put(c.getName(), c);
		animalMap.put(c2.getName(), c2);
		animalMap.put(d.getName(), d);
```

```
animalMap.put(r.getName(), r);
animalMap.put(r2.getName(), r2);
HashMap<Animal, String> animalMap2 = new HashMap<>();
animalMap2.put(c, "Fluffy");
animalMap2.put(c2, "Doesn't like being picked up");
animalMap2.put(d, "Overly excited about running");
animalMap2.put(r, "Snuffles a lot, may have a cold");
animalMap2.put(r2, "May actually be evil. Unsure.");
System.out.println(
       "\nFor each loop on the animal map values ");
for (String key : animalMap.keySet()){
     System.out.println("Key: " + key + " Value: " +
                        animalMap.get(key));
}
// Ex7-2 Task4
HashSet<Animal> set = new HashSet<>();
//we can just add all the elements of another collection
set.addAll(animalList);
set.add(c); // replaces the previous version of the cat
set.add(c2);
// Ex7-2 Task4-3
System.out.println("\nIterator used with the set");
Iterator<Animal> iter = set.iterator();
while(iter.hasNext()){
     System.out.println(iter.next());
}
// Ex7-3 Task1
// this is the same for the animal list and the set, we
// need to iterate through everything
System.out.println("\nFinding Spot.");
for (Animal a : animalList){
    if (a.getName().equals("Spot")){
       System.out.println(
              "Found Spot in the ArrayList: " + a);
     }
}
System.out.println("Found Spot in the HashMap: " +
                   animalMap.get("Spot"));
// Ex7-3 Task2
Collections.sort(animalList);
System.out.println("Sorted animal list");
for (int x = 0; x < animalList.size(); x ++){
     System.out.println(animalList.get(x));
```

```
            }
            // Ex7-3 Task3 - TreeMap example
            TreeMap<String, Animal> tree = new TreeMap<>();
            tree.putAll(animalMap);
            System.out.println("TreeMap");
            for (String key : tree.keySet()){
                System.out.println("Key: " + key + " Value: " +
                                    tree.get(key));
            }
      }
}
```

Animal.java

```
public abstract class Animal implements Comparable<Animal>{
      …
      @Override
      public int compareTo(Animal o) {
            return age - o.getAge();
      }
…
}
```

Lab 8 Exception handling

The objective of this lab is to look at throwing and catching exceptions. By the end of this we should be comfortable with using exceptions and some basic input operations in Java.

Exercise 1 is going to examine the mechanism of the throw clause and the try/catch construct. In Exercise 2 we are going to read input line by line and append each input line to a StringBuffer object. We will then output the contents of StringBuffer object. Console input operations can cause a lot of potential exceptions, so we will need to catch any IOExceptions.

We encourage you to look up how to do specific input operations (which we will look at in detail in the next module), using the Java API or the Oracle tutorials as well as the slides.

Key Terms

Exception

is a runtime error.

StringBuffer

is a mutable class object, which means the objects of StringBuffer can be modified after creation.

BufferedReader

is one of the character stream classes from the Java API which can read data line-by-line.

InputStreamReader

is a bridge class between the byte stream classes and character stream classes.

IOException

is an exception class which is associated with I/O activities.

Exercise 8-1 – Catching and Throwing

Task 1 – Project setup

1. Create a new Java Project, Ex08_Exceptions.
2. Create a package, com.qa.

Task 2 – BadCalc class

1. Create a class, com.qa.BadCalc.
2. Add a method called mult() that takes 2 int parameters, num1 and num2, and returns an int.
3. In the body of the method, return the result of multiplying the 2 parameter values.
4. Add a method called div() that takes 2 int parameters, num1 and num2, and returns an int.
5. In the body of the method, return the result of dividing the 1st parameter by the 2nd parameter.
6. Create a TestCalc class with a main() method.
7. In the body of the main() method, create a BadCalc object and call the 2 methods, printing the results of each.
8. Test your code.
9. Call the div() method with 0 for the num2 argument. You should see an ArithmeticException error. This type of Exception is unchecked and does not need to be handled, but we will handle it to explore the code.

Task 3 – Adding Exception handling

1. In the declaration of the div() method, add a throws clause to let the JVM know that this method may throw an ArthmeticException. This alerts calling code that calls to div() may throw the specified Exception.
2. In the main() method, surround the call to div() with a try/catch construct and output a message and the Exception in the ArithmeticException catch block.
3. Test the code to ensure that you see your message when the Exception occurs, but not when the arguments to div() are OK. This type of error should really be dealt with by ensuring the arguments passed to methods are validated appropriately.

Solution 8-1 – Catching and throwing

Task 2 – BadCalc class

```
package com.qa;
public class BadCalc {
    public int mult(int num1, int num2){
        return num1 * num2;
    }
    public int div(int num1, int num2) {
        return num1 / num2;
    }
}

package com.qa;
public class TestCalc {
    public static void main(String[] args) {
        BadCalc bc = new BadCalc();
        System.out.println("Mult result: " + bc.mult(2, 3));
            System.out.println("Div result: " + bc.div(6, 0));
    }
}
```

Task 3 – Adding Exception handling

```
package com.qa;
public class BadCalc {
    public int mult(int num1, int num2){
        return num1 * num2;
    }
    public int div(int num1, int num2) throws ArithmeticException{
        return num1 / num2;
    }
}

package com.qa;
public class TestCalc {
    public static void main(String[] args) {
        BadCalc bc = new BadCalc();
        System.out.println("Mult result: " + bc.mult(2, 3));
        try {
            System.out.println("Div result: " + bc.div(6, 0));
        } catch (ArithmeticException ae){
            System.out.println("Calculation error caught");
            System.out.println(ae);
            ae.printStackTrace();
        }
    }
}
```

Exercise 8-2 – Exception handling with IO

Task 1 – Read console input

1. Create another class, com.qa.MyConsoleReader, which we will use to gather input lines from the console and append to a StringBuffer object.
2. After the package statement add the following imports to allow us to use classes that will get input from the console (InputStreamReader), process the input line by line (BufferedReader) and handle any input Exceptions (IOException).
```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
```
3. Implement a readAndPrint() method that takes no parameters and returns void.
    a. Create a new StringBuffer object to hold our console data.
    b. We will use the try-with-resources construct to enable automatic closing of our BufferedReader and InputStreamReader, so code the following:
```
try (BufferedReader br = new BufferedReader(
                    new InputStreamReader(System.in));){
}
```
    c. In the body of the try block, prompt for a line of text, then use the BufferedReader's readLine() method to assign a line of console data to a String:
```
String line = br.readLine();
```
    d. We will append this line and subsequent lines to our StringBuffer object until "stop" is entered at the console, so code a while loop that will continue while the String object is not equal to "stop". In the loop:
        i. As the readLine() method removes the newline from a line we will need to add it back in when we append the String to the StringBuffer object, so your append() call will look something like:
```
sb.append(line + "\n");
```
        ii. Prompt for the next line of text.
        iii. Read the next line into your String.
    e. Once the loop terminates, print the value of your StringBuffer object at the end of the try block.
    f. IOException need to be caught for the code to work, so code a catch block for this.

Task 2 – Test console input

4. Code a main method inside MyConsoleReader.
    a. Create an object of the MyConsoleReader class.
    b. Invoke readAndPrint( ) on your MyConsoleReader object.
5. Run your MyConsoleReader, typing lines of input at each prompt on the console, then stop to finish.

Solution 8-2 – Exception handling with IO

Task 1 – Read console input

```
package com.qa;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

public class MyConsoleReader {

    public void readInputPrint() {

        StringBuffer sb = new StringBuffer();

        try (BufferedReader br = new BufferedReader(

                new InputStreamReader(System.in));) {

            System.out.println("Enter a line of text or 'stop'");

            String line = br.readLine();

            while(!line.equals("stop")){

               sb.append(line + "\n");

               System.out.println("Enter a line of text or 'stop'");

               line = br.readLine();

            }

            System.out.println(sb);

        } catch(IOException io){

            System.out.println("Error on console input: " + io);

            io.printStackTrace();

        }

    }

}
```

Task 2 – Test console input

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

        MyConsoleReader mcr = new MyConsoleReader();

        mcr.readInputPrint();

    }
```

Lab 9 File handling

The objective of this exercise is to look at reading and writing operations on files. By the end of this we should be comfortable with some basic file operations in Java.

Use the Java API and the Oracle tutorials as well as the course notes and examples if you need to.

We are first going to read in the lines of a file using a BufferedReader to print the contents. Then we will write data to an output file, and finally read that file in and check the line contents. We will use a File object and later a String to name the files.

Key Terms

StringBuffer

is a mutable class object, which means the objects of type StringBuffer can be modified after creation.

FileReader

is one of the character stream classes from Java API which can refer to the physical location of a file. It is used for implementing reading activities.

BufferedReader

is one of the character stream classes from Java API which can be used to read the data line-by-line. Can be used with a FileReader:

```
BufferedReader br = new BufferedReader(
                          new FileReader("filename"));
```

FileWriter

is one of the character stream classes from Java API which can refer to the physical location of a file. It is used to implement writing operations.

BufferedWriter

is used to write content into a character stream. Can be used with a FileWriter:

```
BufferedWriter bw = new BufferedWriter(
                            new  FileWriter("filename"));
```

Exception

is an error that occurs at runtime.

IOException

is an exception class which is associated with I/O activities.

Exercise 9-1 – Reading from a File

Task 1 – Create a new project, package and class

1. We are going to read in a file line by line using a BufferedReader in order to output the contents to the console. First create a new java project in IntelliJ called Ex09_FileHandling and a package called com.qa.
    a. Add import statements as they are needed.
    b. In Windows you can either use "\\" or "/" as the divider for directories.
2. Create a new Java class called MyFileReader with a main() method and an empty no-arguments constructor.

Task 2 – Process an input file

1. Create a method readAndPrint() that takes no arguments and returns void.
2. Create a new File object passing it the input filename "C:/JAVALabs/Labs/Resources/Sherlock.txt".
3. In a try with resources construct use the File object as the parameter to a new FileReader object which in turn should be used to create a new BufferedReader object.
4. In your try block:
    a. Output a message to introduce the Sherlock file lines.
    b. Read the first line from the BufferedReader into a String variable.
    c. Construct a while loop that will repeat until the line read is null.
        i. Output the current line.
        ii. Read the next line.
5. Now deal with those unhandled exceptions:
    a. Add a catch block for FileNotFoundException, explaining the error in a suitable message with the File details and perhaps issuing a stack trace.
    b. Add a catch block for any other IO exceptions with an appropriate message and possibly a stack trace.
6. Now, in your main() method, create a new MyFileReader object and call your readAndPrint() method on it.
7. Run the application and correct any errors. You should see Project Gutenberg's The Adventures of Sherlock Holmes...
8. Test your exception handling by changing the filename in the File object constructor to a non-existent name.

Solution 9-1 – Reading from a File

```
package com.qa;
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.FileReader;
//Task 1-2
public class MyFileReader {
    public static void main(String[] args) {
        // Task2-6
        MyFileReader mfr = new MyFileReader();
        mfr.readAndPrint();
    }
    public MyFileReader(){
    }
    //Task 2-1
    public void readAndPrint() {
        //Task 2-2
        File myFile = new File(
                          "C:/JAVALabs/Labs/Resources/Sherlock.txt");
        //Task 2-3
        try(BufferedReader br = new BufferedReader(
                                        new FileReader(myFile));) {
            // Task 2-4
            String line = br.readLine();
            while (line != null) {
                System.out.println(line);
                line = br.readLine();
            }
            // Task 2-5
        } catch (FileNotFoundException e) {
            System.out.println("File Not Found: " + myFile);
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Generic Reading Error");
            e.printStackTrace();
        }
    }
}
```

Exercise 9-2 – Writing to a File

Task 1 – Produce an output file

1. Continuing in your MyFileReader class, create a method writeMyFile() that takes no arguments and returns void.
2. Create a String object to hold the name of the output file, "C:/JAVALabs/Labs/Resources/MyOutputFile.txt".
3. In a try with resources construct use the filename String as the parameter to a new FileWriter object which in turn should be used to create a new BufferedWriter object.
4. In your try block:
    a. Output a message to announce that the file is being written.
    b. Build a loop that cycles around 100 times based on a counter starting at 0.
    c. On each cycle append a line to the file:
        i. "This is line: " + counter.
        ii. Include the newline character, "\n".
    d. On completion of the loop, output a message to say that the file writing operation has ended.
5. Now deal with the unhandled exceptions:
    a. Add a catch block for any IO exceptions with an appropriate message and possibly a stack trace.
6. Now, in your main() method, call the writeMyFile() method on your MyFileReader object.
7. Test your application and correct any errors.
    a. You should see your start and end messages displayed.
    b. Look in the Resources directory to confirm that your file has been created.
    c. Open MyOutputFile.txt to check that the lines begin "This is line: " followed by the counter value.

Solution 9-2 – Writing to a File

```
…
import java.io.BufferedWriter;
import java.io.FileWriter;
…
//Ex1 Task 1-2
public class MyFileReader {
    public static void main(String[] args) {
        // Ex1 Task2-6
        MyFileReader mfr = new MyFileReader();
        mfr.readAndPrint();
        // Ex2 Task1-6
        mfr.writeMyFile();
    }
    …
// Ex2 Task1-1
    public void writeMyFile() {
    // Ex2 Task1-2
        String myFileName =
                      "C:/JAVALabs/Labs/Resources/MyOutputFile.txt";
        // Ex2 Task1-3
        try(BufferedWriter bw = new BufferedWriter(
                                    new FileWriter(myFileName));) {
            // Ex2 Task1-4
            System.out.println("Writing to " + myFileName);
            for (int counter = 0; counter < 100; counter++) {
                bw.append("This is line: " + counter + "\n");
            }
            System.out.println("Finished writing to " + myFileName);
        // Ex2 Task1-5
        } catch (IOException e) {
            System.out.println("Caught an IO exception");
             e.printStackTrace();
        }
    }

}
```

Exercise 9-3 – Throwing Exceptions

Task 1 – Read and print your MyOutputFile file

1. We are now going to bring the lines of MyOutputFile.txt back into the application and display them if they are in the correct format. Once that's working, we'll throw an exception if a line is not in the expected format.
2. Continuing in your MyFileReader class, create a method readMyFile() that takes no arguments and returns void.
3. Use a try-with-resources construct to open the file created in Exercise 2 with a BufferedReader and FileReader.
4. In your try block:
    a. Print a header line.
    b. Read in each line from the BufferedReader and if it starts with "This is line", print it out. Look up the startsWith() method in the String class for help if necessary.
5. Catch any unhandled exceptions.
6. In your main() method, call the readMyFile() method on your MyFileReader object.
7. Test your application and correct any errors.
    a. You should see your header line and the lines of the file printed.

Task 2 – Throw your own Exception

1. ... Once that's working, we'll throw an exception of our own if any line is not in the expected format. First create a new class called BadLineException in the same package as your other class.
    a. BadLineException should extend Exception .
    b. We want to store the line of text that was read and considered bad, so create a field to store this text.
    c. Write a getter for the line field.
    d. Write a constructor for the class, it should take in a String message and the line of text that was read when the exception was thrown. The message should be passed to the superclass constructor.
2. Alter the test in readMyFile() so that if the line of text does not start with "This is line" you throw a new BadLineException.
3. Initialise the BadLineException with a message that says what the problem was and the line that we were on.
4. Instead of handling this exception in the try/catch block in the method add "throws" to the method signature for the BadLineException.
5. Now that we are throwing the exception up the call stack to be dealt with

in the calling method, we need to add a try/catch block to that code.

6. In the main method where you call readMyFile(), surround it with a try block and catch BadLineException.
7. Print out the message and the BadLineException line that caused the problem.
8. Test your application.
    a. Initially, the file should be read in successfully.
    b. Alter your code or edit your file to make it contain a line which isn't formatted correctly to get your exception thrown.
    c. Correct any errors.

Task 3 – Streamline readMyFile() - optional

9. Instead of handling some exceptions in readMyFile() and some in main(), amend your code to handle all readMyFile() exceptions in main().

Solution 9-3 – Throwing Exceptions

```
…
public class MyFileReader {
    public static void main(String[] args) {
        // Ex1 Task2-6
        MyFileReader mfr = new MyFileReader();
        mfr.readAndPrint();
        // Ex2 Task1-6
        mfr.writeMyFile();
        // Ex3 Task2-6
        try {
            // Ex3 Task1-6
            mfr.readMyFile();
        } catch (BadLineException e) {
            // Ex3 Task1-7
            System.out.println(e + " - " + e.getTextLine());
        }
    }
    …
    // Ex3 Task1-2          Ex3 Task2-4
    public void readMyFile() throws BadLineException {
        // Ex3 Task1-3
        String myFileName =
                    "C:/JAVALabs/Labs/Resources/MyOutputFile.txt";
        try(BufferedReader br = new BufferedReader(
                                    new FileReader(myFileName));) {
            // Ex3 Task1-4
            System.out.println("Contents of " + myFileName);
            String line = br.readLine();
            while (line != null) {
                if (line.startsWith("This is line")) {
                    System.out.println("Valid Line: " + line);
                    // Ex3 Task2-2
                } else {
                    // Ex3 Task2-3
                    throw new BadLineException("Invalid line found",
                                                line);
                }
                line = br.readLine();
            }
        // Ex3 Task1-5
        } catch (FileNotFoundException e) {
```

```
            System.out.println("File Not Found: " + myFileName);
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Caught an IO exception");
            e.printStackTrace();
        }
    }
}

package com.qa;
// Ex3 Task2-1
public class BadLineException extends Exception{
    private String textLine;
    public BadLineException(String message, String textLine) {
        super(message);
        this.textLine = textLine;
    }
    public String getTextLine() {
        return textLine;
    }
}
```

Lab 10 String handling

By the end of this exercise, you should be able to use a variety of String methods to manipulate data. You will also be able to apply simple regular expressions in the context of the matches() method.

Please feel free to experiment further with the methods as you use them and make use of the JDK API Documentation to explore further methods.

Exercise 10-1 – String methods

Task 1 – Create a new project and class

1. Create a main class LunchMenu in a com.qa package in project Ex10_StringHandling
2. In the main method create an ArrayList of Strings and populate it with a list of about 20 different lunch menu product descriptions.
    a. Include different fillings some with the word sandwich some with roll and some with panini, for example:
        Cucumber sandwich
        Tuna roll
        Feta panini
    b. Copy and paste the elements entered so far and add " with crisps" to some, " with a drink" to some and " with crisps and a drink" to others, for example:
        Cucumber sandwich with crisps
        Tuna roll with a drink
        Feta panini with crisps and a drink
    c. Different flavours of crisps and drink options should also appear separately in the list, for example:
        Cola drink
        Ready salted crisps
3. Print the contents of the list.
4. Test your application so far and correct any errors.

Task 2 – String manipulation

1. Navigate through the ArrayList displaying the descriptions in upper case if the description contains "sandwich", otherwise display it in lower case.
2. Navigate the list printing out all sandwiches.
    a. Print a "Sandwich Options" heading.
    b. Identify where the word sandwich begins (use indexOf()) and display everything from this point to the end of the description so that we can see if the menu option includes a drink or crisps.
    c. Enhance your display by adding a colon (:) after the current display,

followed by the first part of the description up to the word sandwich.

3. Navigate the list printing all paninis in a similar way to step 2c.

Task 3 – Regular expressions

1. List out all menu option descriptions that include both crisps and a drink.
    a. Print a "Full Meal Options" heading.
    b. Use the matches() method to identify the meal options by matching the word "sandwich" or "panini" or "roll", followed by "crisps", followed by "drink" somewhere within the description. Print them out in full.
    c. Enhance your display by omitting the reference to the crisps and drink, just outputting "Cheese roll" or "Hummus panini". Change the heading to include " with crisps and a drink".

Solution 10-1 – String methods

```
package com.qa;

import java.util.ArrayList;

public class LunchMenu {
        public static void main(String[] args) {
        // Ex1 Task1-2 ArrayList populated with lunch options
            ArrayList<String> lunches = new ArrayList<>();
            lunches.add("Salmon and cucumber sandwich");
            lunches.add("Tuna mayonnaise sandwich");
            lunches.add("Cucumber sandwich");
            lunches.add("Tomato sandwich");
            lunches.add("Roast beef sandwich");
            lunches.add("Hummus panini");
            lunches.add("Ready salted crisps");
            lunches.add("Cheese and onion crisps");
            lunches.add("Salt and vinegar crisps");
            lunches.add("Cola drink");
            lunches.add("Orange juice drink");
            lunches.add("Lemon juice drink");
            lunches.add("Bacon, lettuce and tomato sandwich");
            lunches.add(
                  "Salmon and cucumber sandwich with crisps");
            lunches.add(
                  "Salmon and cucumber sandwich with a drink");
            lunches.add(
        "Salmon and cucumber sandwich with crisps and a drink");
            lunches.add("Tuna mayonnaise sandwich with crisps");
            lunches.add(
                  "Tuna mayonnaise sandwich with a drink");
            lunches.add(
               "Tuna mayonnaise sandwich with crisps and a drink");
            lunches.add("Tomato sandwich with crisps");
            lunches.add("Tomato sandwich with a drink");
            lunches.add(
                  "Tomato sandwich with crisps and a drink");
            lunches.add("Cucumber sandwich with crisps");
            lunches.add("Cucumber sandwich with a drink");
            lunches.add(
                  "Cucumber sandwich with crisps and a drink");
            lunches.add("Roast beef sandwich with crisps");
            lunches.add("Roast beef sandwich with a drink");
            lunches.add(
```

```
            "Roast beef sandwich with crisps and a drink");
        lunches.add("Hummus panini with crisps");
        lunches.add("Hummus panini with a drink");
        lunches.add(
              "Hummus panini with crisps and a drink");
        lunches.add("Cheese roll with crisps");
        lunches.add("Cheese roll with a drink");
        lunches.add("Cheese roll with crisps and a drink");
        lunches.add(
          "Bacon, lettuce and tomato sandwich with crisps");
        lunches.add(
         "Bacon, lettuce and tomato sandwich with a drink");
        lunches.add(
"Bacon, lettuce and tomato sandwich with crisps and a drink");
        lunches.add("Feta panini with crisps");
        lunches.add("Feta panini with a drink");
        lunches.add("Feta panini with crisps and a drink");
        // Ex1 Task1-3  Print the list
        System.out.println(lunches);

    // Ex1 Task2-1  Sandwiches in upper case, else lower case
        for(String lunch: lunches) {
            if (lunch.contains("sandwich")) {
                System.out.println(lunch.toUpperCase());
            } else {
                System.out.println(lunch.toLowerCase());
            }
        }
        // Ex1 Task2-2  List of sandwich options
        System.out.println("Sandwich Options");
        for(String lunch: lunches) {
            if (lunch.contains("sandwich")) {
                System.out.println(lunch.substring(
                    lunch.indexOf("sandwich")));
            }
        }

// Ex1 Task2-2c All sandwich options with filling output last
        System.out.println("Sandwich Options");
        for(String lunch: lunches) {
            if (lunch.contains("sandwich")) {
              System.out.println(
              lunch.substring(lunch.indexOf("sandwich"))
              + ": " +
              lunch.substring(0, lunch.indexOf("sandwich"))
```

```
                );
            }
        }

    // Ex1 Task2-3 All panini options with filling output last
        System.out.println("Panini Options");
        for(String lunch: lunches) {
            if (lunch.contains("panini")) {
              System.out.println(
              lunch.substring(lunch.indexOf("panini"))
              + ": " +
              lunch.substring(0, lunch.indexOf("panini"))
              );
            }
        }

        // Ex1 Task3-1 Meal options printed in full
        System.out.println("Full Meal Options");
        for(String lunch: lunches) {
            if (lunch.matches(
               ".*(sandwich|panini|roll).*crisps.*drink")) {
                System.out.println(lunch);
            }
        }
        // Ex1 Task3-1c Meal options omitting extras
        System.out.println(
               "Full Meal Options with crisps and a drink");
        for(String lunch: lunches) {
            if (lunch.matches(
               ".*(sandwich|panini|roll).*crisps.*drink")) {
                System.out.println(
                  lunch.substring(0, lunch.indexOf(" with"))
                );
            }
        }
    }
}
```

Lab 11 Introduction to Lambda

By the end of this exercise, you should be more comfortable with what Lambda expressions are and how to use them in Java.

Exercise 11-1 – Lambda expressions

Task 1 – Create a new project and class

1. Create a new java project in IntelliJ called Ex11_Lambda with a package, com.qa, and a main class called LambdaExample.
2. Create a simple person object:
    a. A person should have a String name, an int age, a double salary and a boolean dueForRetirement.
    b. Implement the constructor, getter, setter and toString() methods for this class.
    c. Implement the Comparable interface and write a compareTo() method for the class which compares people by their name.
3. In the main() method, create an ArrayList of several person objects. Ensure some have dueForRetirement set to true and some of these should have age >= 55.
4. Print the contents of the ArrayList.
5. Remove all Person objects from the ArrayList using the removeIf() method if their dueForRetirement flag is true and their age >= 55.
6. Print the contents of the List to see that the appropriate Person objects have been removed.
7. Sort the ArrayList using the Collections.sort() method. By default, it uses the compareTo() method of the elements. Display the contents noting the ascending name order.
8. Sort the ArrayList using List's sort() method with the compareTo() method in the Lambda Comparator argument. Display the contents again noting the ascending name order.

If you have time:

9. Sort the List again based on ages and provide the sort() method with a Lambda expression for the Comparator argument: (p1, p2) -> p1.getAge() – p2.getAge().
10. Print the list to confirm the Person objects are now in ascending age order.

For an extra challenge:

11. Award each person a 10% increase in salary using the replaceAll() List method with a Lambda argument.
12. Print the contents of the ArrayList to confirm the salary increase.

Solution 11-1 – Lambda expressions

```
package com.qa;

import java.util.ArrayList;
import java.util.Collections;

// Ex1 Task 1-1
public class LambdaExample {
    public static void main(String[] args) {
        // Ex1 Task1-3
        ArrayList<Person> personList = new ArrayList<Person>();
        personList.add(new Person("Eve", 19, 10000, false));
        personList.add(new Person("Alice", 24, 15000, false));
        personList.add(new Person("Dave", 27, 48000, false));
        personList.add(new Person("Mallory", 57, 18000, false));
        personList.add(new Person("Ami", 50, 50000, true));
        personList.add(new Person("Bob", 39, 17000, false));
        personList.add(new Person("Chris", 62, 50000, true));
        personList.add(new Person("Fran", 39, 17000, false));
        // Ex1 Task1-4
        System.out.println("Initial List:\n" + personList);
        // Ex1 Task1-5
        personList.removeIf(p ->
                (p.isDueForRetirement() && p.getAge() >= 55));
        // Ex1 Task1-6
        System.out.println(" without retirees:\n" + personList);
        // Ex1 Task1-7
        Collections.sort(personList);
        System.out.println(
                "Sorted by name using Collections.sort():\n"
                                                + personList);
        // Ex1 Task1-8
        personList.sort((p1, p2) -> p1.compareTo(p2));
        System.out.println("Sorted by name using List's sort():\n"
                                                + personList);
        // Ex1 Task1-9
        personList.sort((p1, p2) -> p1.getAge()-p2.getAge());
        // Ex1 Task1-10
        System.out.println("Sorted by age using List's sort():\n"
                                                + personList);
        // Ex1 Task1-11
        personList.replaceAll(p -> new Person(p.getName(),
                            p.getAge(),
```

```
                                p.getSalary() * 1.1,
                                p.isDueForRetirement()));
        // Ex1 Task1-12
        System.out.println("After salary increases:\n"
                                                    + personList);
    }
}
```

Lab 12 Streams

The aim of this exercise is to use functional programming concepts and by the end of you should be more comfortable with Lambda expressions and streams and how to use them in Java.

With all these exercises, it will be possible to find a non-functional way of achieving the same result. If it helps, try thinking about the standard object-oriented approach first and then change it around, but remember that the aim of this exercise is to use functional programming concepts!

You may wish to comment out earlier parts of the lab when they are not needed for a later step.

Key Terms

Lambda Expressions

are the same concept as anonymous functions.

ArrayList

is one of the implementation classes of the List interface and it is not synchronised.

Streams

are an abstract representation of data flow, where functional methods are available to streams that are not available to the standard collection classes.

Exercise 12-1 – Lambda expressions

Task 1 – Create a new project, class and Stream of Integers

1. Create a new java project in IntelliJ called Ex12_Streams with a package, com.qa, and a main class, StreamsExample.
2. In the main() method, use a range method to create an Integer Stream of years from 1960 to 2050.
3. Using a chain of operations on the Stream to do the following:
4. Filter this list to extract leap years use the modulo operation %. A year is a leap year if it:
    a. is divisible exactly by four AND
    b. is not exactly divisible by 100 OR
    c. is exactly divisible by 400
5. Print out each element of the stream.

Task 2 – Create a Person class and ArrayList of Persons

1. Create a Person class:
    a. A person should have a String name, an int age, and a double salary.
    b. Implement a constructor for all fields, getter, setter and toString() methods for this class.
    c. Implement the Comparable interface and write a compareTo() method for the class which compares people by their name.
2. In the main() method, create an ArrayList called personList of several Person objects.
3. Display the list.

Task 3 – Chained operations on a Stream of Persons

1. Create a Stream of elements from the Person list to:
    a. Filter the Person stream for people over the age of 30 who have a salary of less than 20,000.
    b. Print out each element of the stream.
2. Create a Stream of elements from the Person list to:
    a. Filter the Person stream for people over the age of 30 who have a salary of less than 20,000.
    b. Map the Person elements to String elements containing just the name.
    c. Print out each name from the stream.
3. Create a Stream of elements from the Person list to:
    a. Filter the Person stream for people over the age of 30 who have a salary of less than 20,000.
    b. Map the Person elements to String elements containing just the name.
    c. Use Collectors.toList() in the stream's collect() method to create a List of Strings, namesList, from the stream.
    d. Display the new List.
4. Create a Stream of elements from the Person list to:
    a. Filter the Person stream for people over the age of 30 who have a salary of less than 20,000.
    b. Use map() to increase the filtered person's salary by 10%. This will create a stream of new Person objects with the increased salary, to be passed to the next chained operation.
    c. Sort the stream elements using the Streams sorted() method.
    d. Print out each Person from the stream.

e. What happens if you remove the sorted method()? What happens if you try and change the collection after the print statement has run?

5. Create a Stream of elements from the Person list to:
    a. Filter the Person stream for people over the age of 30 who have a salary of less than 20,000.
    b. Use map() to increase the filtered person's salary by 10%.
    c. Sort the stream elements using the streams sorted() method.
    d. Use collect() to create a new List of Person objects, incSalPersonList.
    e. Print the incSalPersonList.
6. Get Summary Statistics as a String for salaries of people under 30:
    a. Create a Stream of Person objects from personList.
    b. Filter those with age less than 30.
    c. Use collect() with the Collectors method summarizingDouble() giving it the salaries of Person objects in the stream.
    d. Use the stream's toString() method to return the summary statistics as a String, summaryStats.
    e. Print summaryStats.
7. Calculate the average salary of people under 30 into a double:
    a. Create a Stream of Person objects from personList.
    b. Filter those with age less than 30.
    c. Use collect() with the Collectors method summarizingDouble to get the salary statistics.
    d. Use getAverage() to extract the average salary into a double avgSalUnder30.
    e. Print the rounded avgSalUnder30 value.
8. Print the average salary of people over 30:
    a. In System.out.println(), start with "Average salary of those over 30 is ".
    b. Concatenate onto that the rounded result of getting the average salary of those people who are over 30.

Solution 12-1 – Lambda expressions

StreamsExample.java

```
package com.qa;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
// Ex12-1  Task1-1
public class StreamsExample {
    public static void main(String[] args) {
        // Ex12-1  Task1-2
        IntStream years = IntStream.range(1960, 2050);
        // Ex12-1  Task1-4
        years.filter(y -> (y % 4 == 0) && (y % 100 != 0)
                                       || (y % 400 == 0))
              // Ex12-1  Task1-2
             .forEach(System.out::println);
        // Ex12-1  Task2-2
        ArrayList<Person> personList = new ArrayList<>();
        personList.add(new Person("Eve", 19, 10000));
        personList.add(new Person("Alice", 24, 15000));
        personList.add(new Person("Dave", 27, 48000));
        personList.add(new Person("Mallory", 42, 18000));
        personList.add(new Person("Ami", 50, 50000));
        personList.add(new Person("Bob", 39, 17000));
        personList.add(new Person("Chris", 31, 19000));
        // Ex12-1  Task2-3
        System.out.println(personList);
        // Ex12-1  Task3-1
        personList.stream()
                  .filter(p -> p.getAge() > 30
                            && p.getSalary() < 20000)
                  .forEach(System.out::println);
        // Ex12-1  Task3-2
        personList.stream()
                  .filter(p -> p.getAge() > 30
                            && p.getSalary() < 20000)
                  .map(p -> p.getName())
                  .forEach(System.out::println);
        // Ex12-1  Task3-3
```

```
List<String> namesList =
    personList.stream()
              .filter(p -> p.getAge() > 30
                        && p.getSalary() < 20000)
              .map(p -> p.getName())
              .collect(Collectors.toList());
System.out.println("New List of Names: " + namesList);
// Ex12-1  Task3-4
personList.stream()
          .filter(p -> p.getAge() > 30
                    && p.getSalary() < 20000)
          .map(p -> new Person(p.getName(), p.getAge(),
                               p.getSalary() * 1.10))
          .sorted()
          .forEach(System.out::println);
// Ex12-1  Task3-5
List<Person> incSalPersonList = personList
          .stream()
          .filter(p -> p.getAge() > 30
                    && p.getSalary() < 20000)
          .map(p -> new Person(p.getName(), p.getAge(),
                               p.getSalary() * 1.10))
          .sorted()
          .collect(Collectors.toList());
System.out.println(incSalPersonList);
// Ex12-1  Task3-6
String summaryStats = personList
        .stream()
        .filter(p -> p.getAge() < 30)
        .collect(Collectors.summarizingDouble(
                                  p -> p.getSalary()))
        .toString();
System.out.println("Summary statistics of those under 30: "
                                        + summaryStats);
// Ex12-1  Task3-7
double avgSalUnder30 = personList
          .stream()
          .filter(p -> p.getAge() < 30)
          .collect(Collectors.summarizingDouble(
                                     p -> p.getSalary()))
          .getAverage();
System.out.println("Average salary of those under 30 is "
                           + Math.round(avgSalUnder30));
// Ex12-1  Task3-8
```

```
        System.out.println("Average salary of those over 30 is "
          + Math.round(personList
                .stream()
                .filter(p -> p.getAge() > 30)
                .collect(Collectors.summarizingDouble(
                                        p -> p.getSalary()))
                .getAverage())
        );
    }
}
```

Person.java

```
package com.qa;
// Ex12-1 Task2-1
public class Person implements Comparable<Person>{
    private String name;
    private int age;
    private double salary;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public double getSalary() {
        return salary;
    }


    public void setSalary(double salary) {
        this.salary = salary;
    }

    public Person(String name, int age, double salary) {
        super();
        this.name = name;
```

```
        this.age = age;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Person [name=" + name + ", age=" + age +
                        ", salary=" + salary + "]";
    }

    @Override
    public int compareTo(Person p) {
        return name.compareTo(p.getName());
    }


}
```

Exercise 12-2 – Lambda parameter

Task 1 – Use a method with a Lambda parameter

1. Create a static method applyFunction() that
    a. Takes an ArrayList of Integers and a Lambda Function<Integer, Integer> as its two parameters
    b. Returns an ArrayList of Integers.
2. In the applyFunction() method, return a new ArrayList constructed from:
    a. A stream created from the input ArrayList.
    b. Manipulated by passing the input function to map().
    c. Collected into a List.
3. In the main() method create an ArrayList of Integers, intList, and assign it a new ArrayList() giving the constructor:
    a. A Stream of ints with values 0 to 10 by using IntStream.range().
    b. Converted to an Integer Stream using .boxed().
    c. Collected into a List.
4. Print intList.
5. Call the applyFunction() method to return an ArrayList of Integers called oneUp by passing it intList and the Lambda Function i -> i = i+ 1.
6. Print oneUp and notice the difference between that and intList.
7. Call the applyFunction() method to return an ArrayList of Integers called doubleUp by passing it intList and a Lambda Function to double the element values.
8. Print doubleUp and check your results.
9. Call the applyFunction() method to return an ArrayList of Integers called squares by passing it intList and a Lambda Function to square the element values.
10. Print squares and check your results.

Solution 12-2 – Lambda parameter

```
…
public class StreamsExample {
    public static void main(String[] args) {
      …
        // Ex12-2  Task1-3
        ArrayList<Integer> intList = new ArrayList<>(
                            IntStream.range(0, 10)
                                     .boxed()
                                     .collect(Collectors.toList()));
        // Ex12-2  Task1-4
        System.out.println(intList);
        // Ex12-2  Task1-5
        ArrayList<Integer> oneUp =
                             applyFunction(intList, i -> i = i + 1);
        // Ex12-2  Task1-6
        System.out.println(oneUp);
        // Ex12-2  Task1-7
        ArrayList<Integer> doubleUp =
                             applyFunction(intList, i -> i = i * 2);
        // Ex12-2  Task1-8
        System.out.println(doubleUp);
        // Ex12-2  Task1-9
        ArrayList<Integer> squares =
                             applyFunction(intList, i -> i = i * i);
        // Ex12-2  Task1-10
        System.out.println(squares);
    }

    // Ex12-2  Task1-1
    public static ArrayList<Integer> applyFunction(
                                      ArrayList<Integer> input,
                                      Function<Integer, Integer> f) {
        // Ex12-2  Task1-2
      return new ArrayList<Integer>(
                                input.stream()
                                     .map(f)
                                     .collect(Collectors.toList()));
    }
```

Lab 13 Packaging

By the end of this exercise, you should be able to generate jar files from your source code using IntelliJ.

Exercise 13-1 – Package an application into a JAR file

Task 1 – Create and run a JAR file

1. Choose one of your projects to prepare for distribution (Ex5_Inheritance possibly).
2. Generate a jar file for it using IntelliJ. First create the Artifact:
    a. File -> Project Structure -> Artifacts (under Project Settings).
    b. Click + and select JAR -> From modules...
    c. Select the required main class and click extract to target JAR.
    d. Ensure the appropriate src directory is selected for the manifest.
    e. Click OK on Create JAR from Modules dialog.
    f. Click OK on Project Structure.
3. Now build the Artifact:
    a. Build -> Build Artifacts -> projectname:jar -> Build
    b. In the Project navigator you can expand the out folder and artifacts folder to see your .jar file.
    c. Right click your .jar file and Run projectname.jar.
4. Run the jar file from the command line (optional).
    a. Open a command line session (cmd).
    b. Navigate to the directory containing your JAR file.
    c. Issue the command:

```
java -jar projectname.jar
```

Lab 14 Documentation

The objective is to look at how to write Javadoc and generate some for our own classes. Use the classes created in Ex05_Inheritance, the Objects exercise.

Exercise 14-1 – Generating JavaDoc

Task 1 – Add JavaDoc Comments

1. Open your classes from Ex05_Inheritance and write some Javadoc comments for the classes and the methods using /** ... */. Describe the functionality for each of the methods.
2. Generate the Javadoc from these comments in IntelliJ.
    a. Tools -> Generate JavaDoc.
    b. In the Generate JavaDoc dialog, select Whole project and choose a dedicated directory to hold the generated documentation, for example, docs in the project directory. It can create this directory for you. Click Generate.
3. Have a look at the generated documentation. Do the comments reflect what you wanted to explain? Notice that the format is identical to that of the Java API documentation.

Lab 15 Test Driven Development

This exercise looks at how to use TDD and JUnit as well as write a test case on a POJO (Plain-Old-Java-Object) class. By the end of the exercise, you should be more comfortable with TDD and JUnit.

Key Terms

Object

is the combination of state and behaviour.

Programming object

is an instance of a class.

Class

is a specification of a real-world object, acts as a placeholder for variables and methods.

Constructor method

is a special member method which is called when the Java class is instantiated.

Method

is a block of code which can be declared once and called many times.

Instantiation

is the process of creating objects from the class.

Unit testing

involves a set of tests for a unit that can be a method, database query, stored procedure, transaction, or a dynamic web page.

Test Driven Development

is a core of XP (eXtream Programming) and can be adopted within other methodologies. It relies on three main factors: Red-Green-Refactor.

Exercise 15-1 – JUnit test

Task 1 – Project setup and creating a Class Under Test (CUT)

1. Create a new java project in IntelliJ:
    a. Name it Ex15_TDD
    b. Language: Java
    c. Ensure that in the New Project dialog you select:
    Build system: Maven
    d. Click Create.
2. For our project to use JUnit features, we need to add JUnit as a dependency:
    a. Open **pom.xml** in the root directory of your project.
    b. Right click and select **Generate** -> **Add dependency.**
    c. At the bottom of the IntelliJ screen in the Dependencies tool window, under Dependencies: Manage, type `org.junit.jupiter:junit-jupiter` in the search field.
    d. Locate the **Junit Jupiter(Aggregator)** dependency in the search results and click **Add** next to it. The dependency is added to your pom.xml.
    e. Click **Load Maven Changes** in the notification that appears in the top-right corner of the editor to apply the changes in the build script.
3. In the Project tool window, go to **src/main/java** and create a com.qa package.
4. Create an Account class.
    a. It should have fields:
        i. cardTypes (array of String objects)
        ii. sortcode (String)
        iii. accountNumber (String)
        iv. password (String)
    b. It should implement the Comparable<Account> interface.
5. Write a constructor that sets all the fields in the class.
6. Create getter and setter methods for each of the fields in the class.
7. Create a **toString()** method by concatenating all fields in a readable manner.
8. Override the **compareTo()** method.

a. If the first cardTypes entry of the other Account matches any of this Account's cardTypes and all other fields are identical return 0.
`if(Arrays.asList(this.cardTypes).contains(other.cardTypes[0]))...`

b. If the first cardTypes entry of the other Account doesn't match any of this Account's cardTypes or any of the other fields differ return -1.

The above steps conclude the creation of Class Under Test.

Task 2 – Creating a JUnit test

1. Right-click on the Account class declaration and select Show Context Actions -> Create Test.
2. In the Create Test dialog:
   a. Give the Class name: AccountTest
   b. Destination package: com.qa
   c. Generate setUp/@Before
   d. Under Member check compareTo()
   e. Click OK.
3. The AccountTest class is generated for us containing:
   a. A setUp() method with the @BeforeEach annotation in which to code actions prior to the test.
   b. A compareTo() method with the @Test annotation in which to code the test details.
4. In the AccountTest class, declare a private variable with Account class type:

```
private Account act;
```

5. In the setUp() method, create a new Account object, supplying values for all fields to the constructor:

```
act = new Account(new String[]{"Link","Visa"},
                  "10-10-22","11223344","1234");
```

6. In the test compareTo () method, create an object from Account class using sample input values:

```
Account inputObject = new Account(new String[]{"Link"},
                      "10-10-22","11223344","1234");
```

7. Invoke assertEquals() by sending the corresponding arguments:

```
assertEquals(0, act.compareTo(inputObject),
             "Login Failed");
```

8. Just before the compareTo() method add the @DisplayName annotation

to give the test a meaningful descriptive name:
`@DisplayName("Match Accounts")`

9. Execute a JUnit Test by clicking the green triangle in the gutter next to the test and selecting Run, or click the green triangle next to the AccountTest class declaration and select Run to execute all tests.
10. Results of the tests appear at the bottom left of the screen with a green tick for success or a cross on a yellow background for Assertion failure.
11. Devise further tests for some other Account methods and execute them.

Solution 15-1 – JUnit test

```
package com.qa;
import java.util.Arrays;
public class Account implements Comparable<Account>{
    private String cardTypes[];
    private String sortCode;
    private String accountNumber;
    private String password;
    public Account(String[] cardTypes, String sortCode,
                   String accountNumber, String password) {
        super();
        this.cardTypes = cardTypes;
        this.sortCode = sortCode;
        this.accountNumber = accountNumber;
        this.password = password;
    }
    public String[] getCardTypes() {
        return cardTypes;
    }
    public void setCardTypes(String[] cardTypes) {
        this.cardTypes = cardTypes;
    }
    public String getSortCode() {
        return sortCode;
    }
    public void setSortCode(String sortCode) {
        this.sortCode = sortCode;
    }
    public String getAccountNumber() {
        return accountNumber;
    }
    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
@Override
public int compareTo(Account thatAct) {
```

```
if(Arrays.asList(this.cardTypes).contains(thatAct.cardTypes[0]))
            if(this.sortCode.equals(thatAct.sortCode))
                if(this.accountNumber.equals(thatAct.accountNumber))
                    if(this.password.equals(thatAct.password))
                    {
                        return 0;  // Login Success
                    }
        return -1;  // Login fails
    }
}
```

```
package com.qa;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class AccountTest {
    private Account act;
    @BeforeEach
    void setUp() {
        // Create an Account
        act = new Account(new String[]{"Link","Visa"},
                          "10-10-22","11223344","1234");
    }
    @Test
    @DisplayName("Match Accounts")
    void compareTo() {
        Account inputObject = new Account(new String[]{"Link"},
                                  "10-10-22","11223344","1234");
        assertEquals(0, act.compareTo(inputObject), "Login Failed");
    }
    @Test
    @DisplayName("Check Account Number")
    void getAccountNumber() {
        assertEquals("11223344", act.getAccountNumber(),
                     "Account Number mismatch");
    }
}
```

Lab 16 Introduction to Java Modules

There is no exercise for this section.

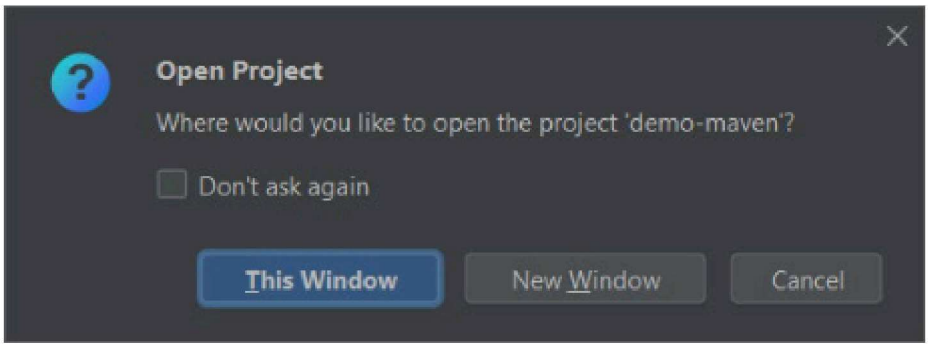
Lab 17 Maven introduction

The aim of this exercise is to create and run a basic Maven project. You will work with the POM and Maven repositories, perform a unit test with Maven and show a unit testing report.

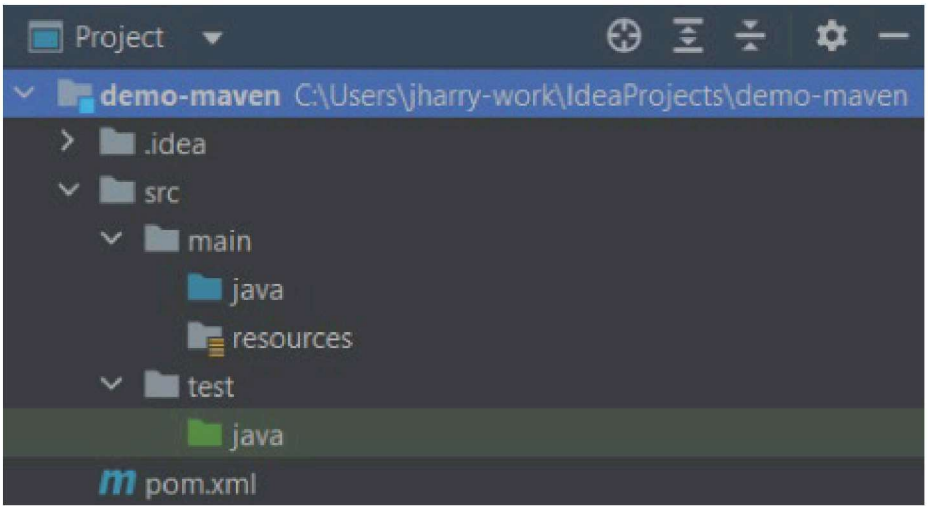
Exercise 17-1 – Maven hello world project

Task 1 - Creating a new Maven project

1. Launch IntelliJ.
2. Go to **File** > **New** > **Project...**
3. **Create** a project called demo-maven and make sure Maven is selected as the **build system.**
4. Select **This Window** when asked where you would like to open the project.



5. Look at the structure of the created project:



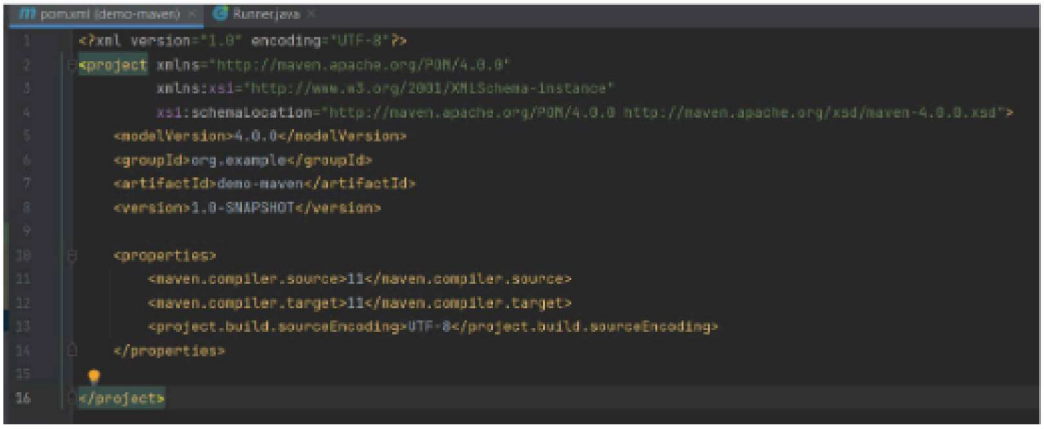
6. Try to remember from the presentation what the purpose of each file/folder is. Can you remember them all?

Task 2 - Running a Hello World Maven project

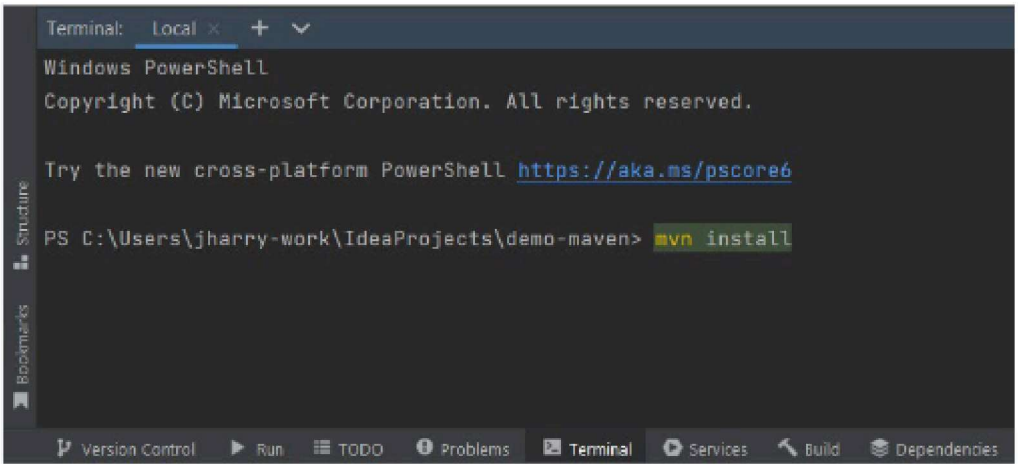
1. In the demo-maven project, you created earlier create a new package called com.qa.
2. Inside this new package create a Runner class.
3. Add a main method to the Runner class.
4. Make the main method print out "Hello, World!".
5. Run your program to ensure it's set up correctly.

Task 3 - Working with the POM

1. If you open the **POM** file in your project, you should see something that looks like this:



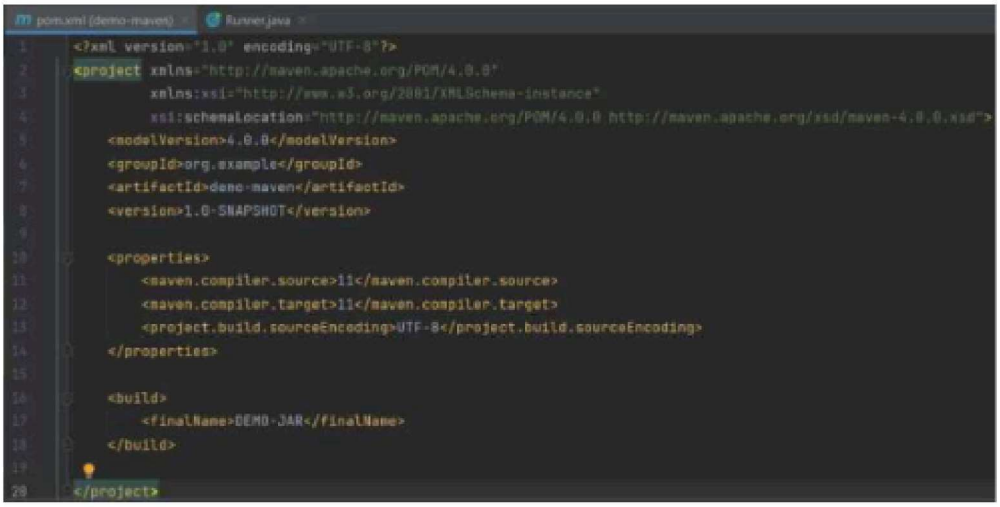
2. Try and remember what each section of the POM means – **groupid**, **artifactid**, **version**, **properties**, and the other tags you may see.
3. In a moment you're going to modify this file but for now build this project to a jar file using **mvn install.**



4. If this succeeds you should see a jar file in the target folder of your project and **BUILD SUCCESS** in the terminal.
5. This name is fine, but a bit unwieldy. Happily, we can change it using

the POM file. Go back to your **POM** file and add a **build** tag after **properties** but before **</project>**.

6. **In the** build tag add a **finalName** tag and write Demo-Jar inside it:



7. Now remove your previous build using **mvn clean.**

8. Rebuild your jar file using **mvn install** again. You should see the jar file has the new name from the POM file.

9. Now that we've seen the **build** tag let's start working with plugins. Most jar files are only used for importing into other projects as a dependency, but in some cases, it can be useful to be able to run our jars.

   A jar file must be made executable to be run. We will do this by using the **maven-jar** plugin. Add the plugin to your project by inserting this snippet into your **build** tag (preferably between </finalName> and </build>):

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <configuration>
            <archive>
                <manifest>
                </manifest>
            </archive>
        </configuration>
    </plugin>
</plugins>
```

10. The only thing left to configure now is the **manifest.** In Java, the manifest is a file that contains metadata about a jar file.

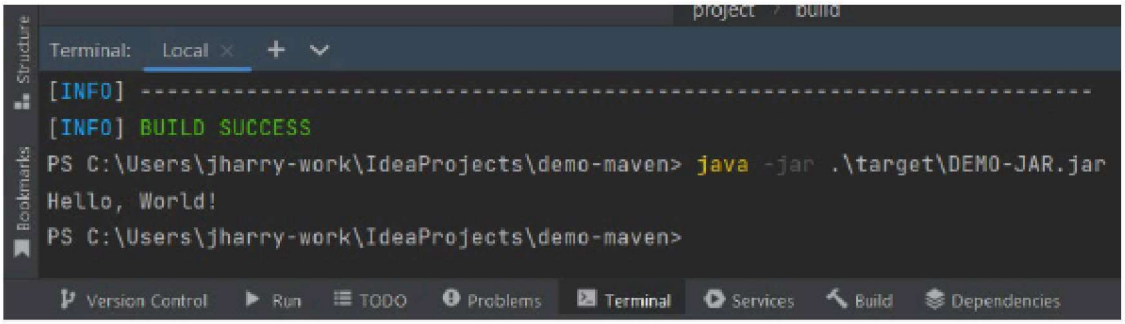
    To make your jar executable, you will need to add the **mainClass** to your manifest – this will allow you to tell Java which class contains the main method using its full name.

```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <configuration>
            <archive>
                <manifest>
                    <mainClass>
                        com.qa.Runner
                    </mainClass>
                </manifest>
            </archive>
        </configuration>
    </plugin>
</plugins>
```

11. Rebuild your project using **mvn clean install.** Although they won't prevent your build from succeeding, plugin version warnings can be eliminated by including the following line after </artifactId> and before <configuration>:

    <version>3.3.0</version>

12. Try executing your jar file – do you see a build success in your terminal?



Exercise 17-2 – Using dependencies

Task 1 - Working with the POM

1. Start by opening the maven-testing – start project in your IDE. This project contains a very basic calculator with some simple tests.

2. At the moment, we have the code and the tests, but the JUnit library isn't imported so the first thing to do is add it as a dependency. Open the POM file and add a **dependencies** tag after the **properties** tag.

3. In the **dependencies** tag add a **dependency** with a **groupid**, **artifactid**, **version**, and **scope**.

```
<dependencies>
    <dependency>
        <groupId></groupId>
        <artifactId></artifactId>
        <version></version>
        <scope></scope>
    </dependency>
</dependencies>
```

4. The dependency we want to import is JUnit so enter **junit** as the **groupId** and **artifactId**, set the **version** as 4.13.2 and the **scope** as test.

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

5. Now try running a **clean install** – you should see maven running the tests as it attempts to build the project.

6. At this point, the build will fail due to an incorrect test (**testDivide**). If you check the target folder, you should see that, whilst there are several generated files from the last build, *there is no jar file.*

This is because our build process failed at the **test** stage and therefore never reached the **package** stage. In industry, this is an important role of Continuous Delivery/Deployment as it makes it much harder to package up bad code.

7. Fix the **testDivide()** method in **CalcTest** by adding 0.1 as the third parameter for the **assertEquals** (previously the check was far too specific, by adding a delta value of 0.1 the test will just check for an approximate number and the test should pass).

```
@Test
public void testDivide() {
    Assert.assertEquals(3.33333, Calc.divide( a: 10,  b: 3), 0.1);
}
```

8. Run another **clean install** – this time the build should succeed, and you should see a jar file in the target **folder.**

Task 5 - Generating a Surefire Report

1. Up till now, Maven will have been generating **surefire reports** for each of the test classes and storing them in the target folder.

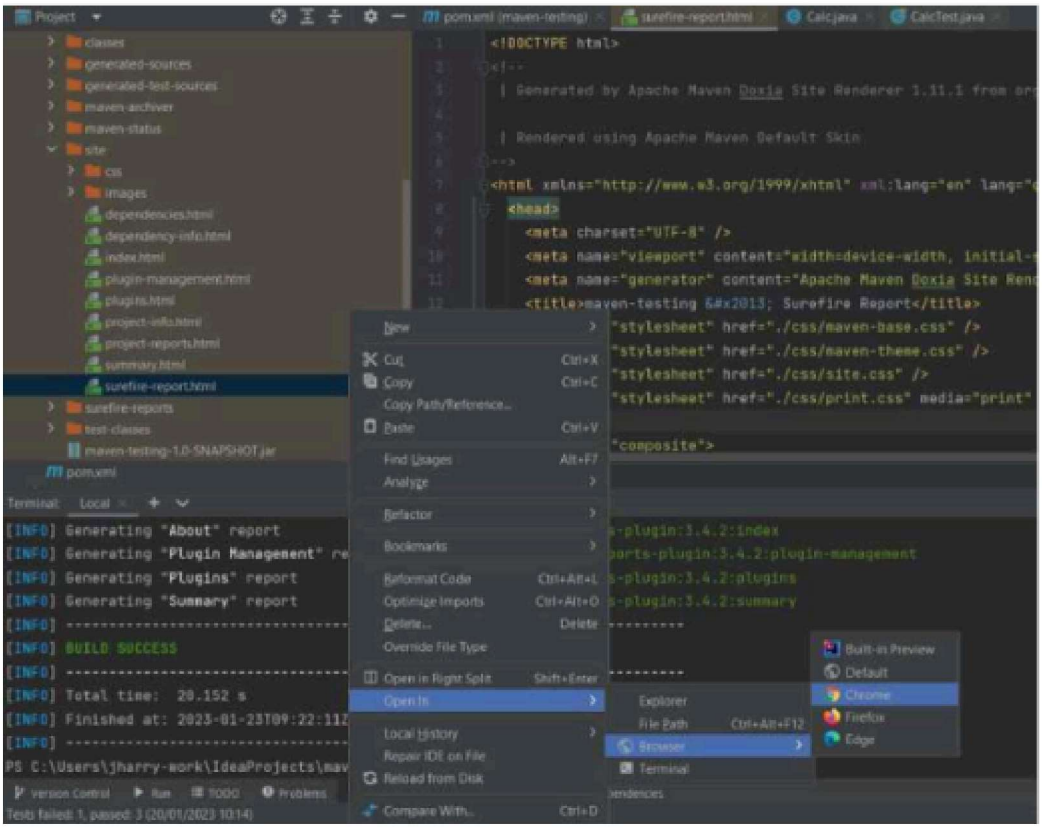
   If we wish to make these reports much more readable, we can add the Surefire-Report plugin to our project and display the test results in a simple webpage. Start by adding this code snippet to your POM file, but with **`<version>3.1.2</version>`.**

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-report-plugin</artifactId>
            <version>3.0.0-M8</version>
        </plugin>
    </plugins>
</reporting>
```

2. You can view the original report from the **target** folder in your IDE after a **mvn install.**

3. To generate the readable report, run **mvn site.**

4. Open the report in a web browser of your choice.



5. Look through the webpage; you should see a summary of all the tests that were run, whether they passed or failed and the time they took to run.