Programming IT, Laboratory Exercises

Book 1: introductory week

Dr. Simon Rogers & Prof. Alice Miller
January 2021

How to use these exercises:

- These exercises are for you to work through at your own pace. The order of material follows that of the lectures, but don't feel that you have to do particular exercises on particular days: do whatever works best for you. Academics and lab assistants will be happy to help with any of this material at any time.
- Don't feel restricted to these exercises. A great way to learn to program is to set yourself your own exercises. Think of a problem or task and then try and write a program to solve it. We will be happy to help. If that feels too daunting, start with one of these exercises and play around: change something, predict what will happen, and then see if it does!
- Some of the exercises mirror the work we have done in the lectures. If you're confident that you got it in the lecture, skip them. If you're not sure, do them for yourself.

Some general advice:

- Part of the challenge of learning to program is learning to think like a programmer. This can take a while: don't panic!
- When you're a beginner faced with a programming problem, don't fall into the habit of always jumping straight into writing code. Sketch out the solution on paper first (maybe a flowchart will help).
- As well as writing code, reading code is an excellent way to learn. Read code, and try and predict what it's doing. Then run it, and see if you were right!
- Eclipse is useful, but try not to become too reliant on it. You'll learn a lot if you use other tools sometimes (e.g. a text editor plus a compiler).
- It's easy to overlook testing. When you're writing a program, think of how you will convince yourself that it works by thinking up tricky test cases.

1 Introduction

These exercises are to help you become more familiar with *compiling* and *running* Java programmes.

1.0 Using the UofG remote desktop

UofG have provided a remote desktop that can be accessed through a web browser.
Pros: no installation or eclipse / Java required; work backed up (if you set your workspace within your one drive folder).
Cons: a bit slower than using eclipse on your own machine; have to use Eclipse (no other IDE installed)

To access the remote desktop, go to:
https://rdweb.wvd.microsoft.com/arm/webclient/index.html
Scroll down to “College of Science and Engineering” (or whatever is correct for you)
Login with your GUID and password
Voila!
You will find Eclipse in the finder menu.

Below are instructions for setting up your Eclipse Workspace. This is where Eclipse stores your files. We **strongly** recommend that you make a folder in your one drive which will be backed up, and available anywhere. To do this:

- The first time you login, double click on the onedrive folder on the desktop. You might need to login.
- You should now see onedrive in the finder menu. Add a folder for your eclipse workspace (convention says to name it eclipse-workspace)
- When you start eclipse, choose this folder as the workspace

1.1 Hello World

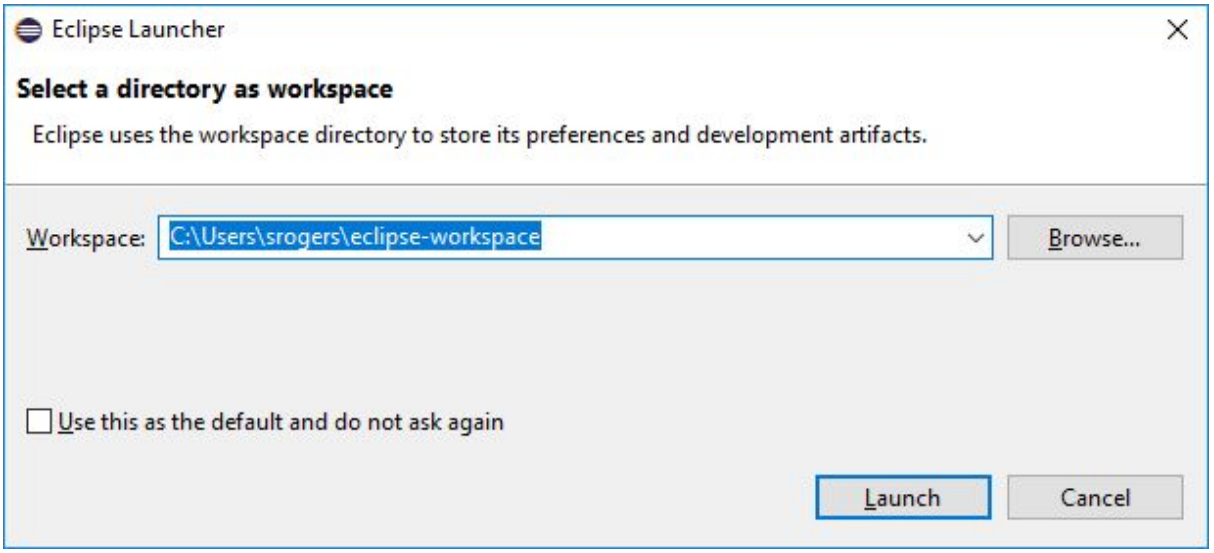
In this exercise you will write a “Hello World” programme using the Eclipse IDE. Keep in mind that a lot of these tasks are Eclipse-specific and not really “programming”!

Note: Your Eclipse workspace is where Eclipse stores the programmes you write. If you are on campus and using a University machine it is **very important** that this is set to somewhere on your network (H:) drive. If it isn’t, Eclipse will store the programmes locally on the computer you are using in the lab and you won’t be able to access them at a different computer.

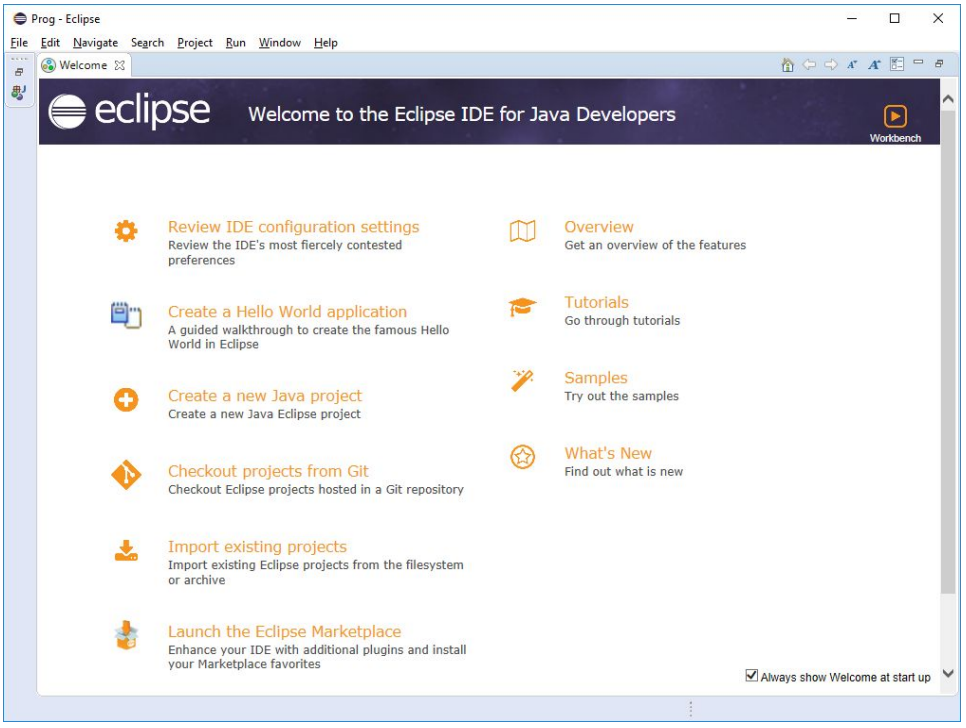
Backup your code regularly!

1) Open Eclipse (the easiest way is to click on the magnifying glass towards the bottom left of the screen and then type Eclipse)

2) When you open for the first time, you will be prompted to set your workspace (it depends if you have opened Eclipse before):

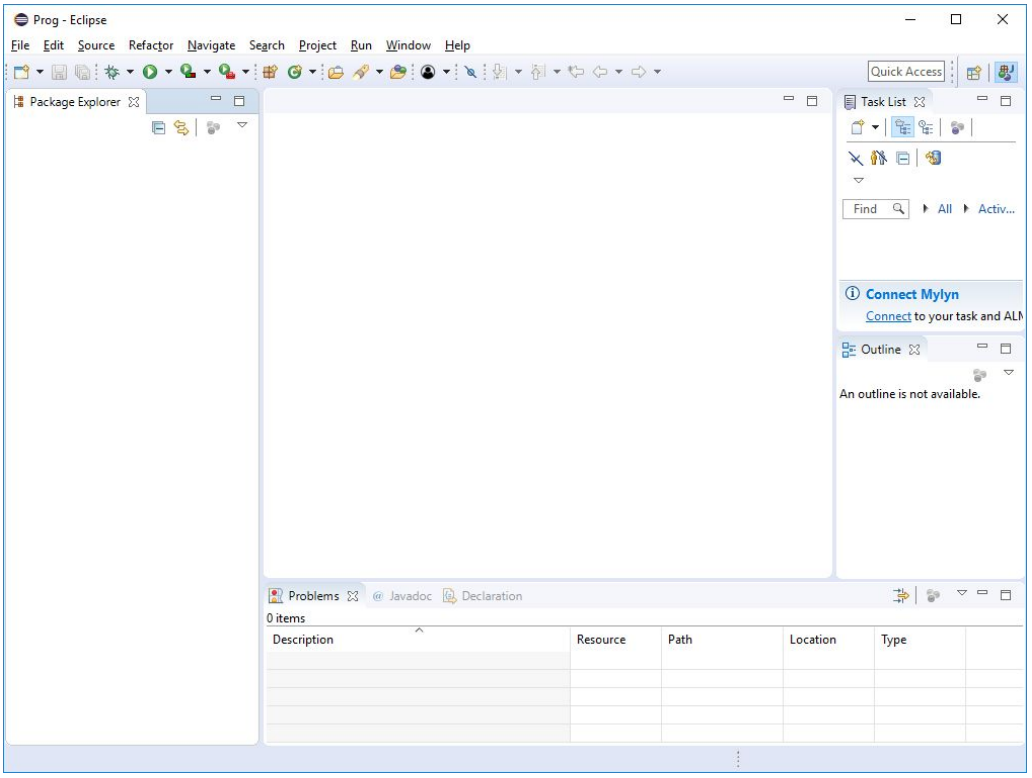


3) By default, Eclipse uses the ‘C’ drive, which is the hard disk on the particular computer you are using.
4) At any point you can switch workspaces by choosing “File->Switch Workspace”.
5) Eclipse will now give you a nice Welcome screen:



6) Close it by clicking on the ‘x’ next to the word “Welcome” in the top left.

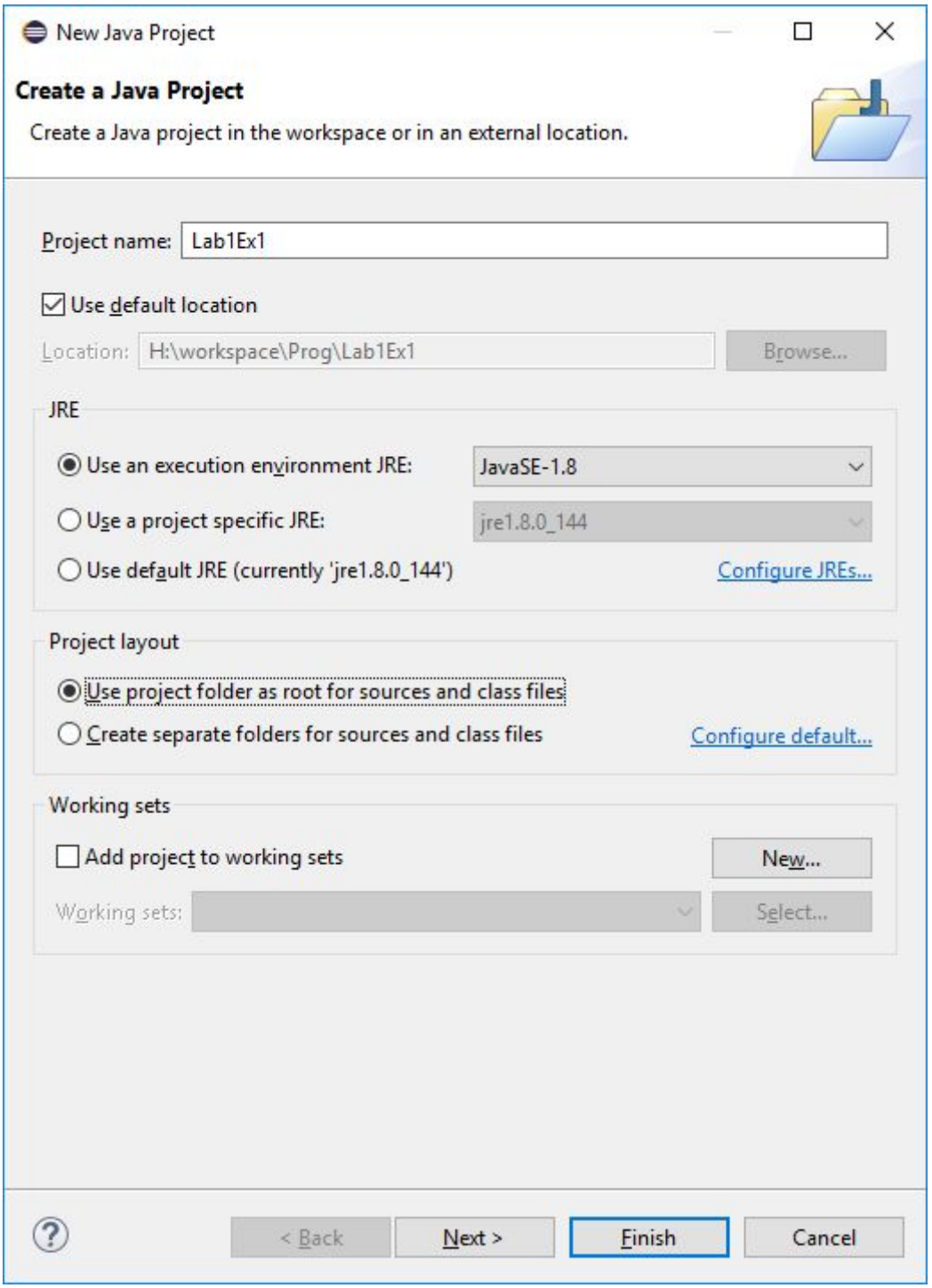
7) You will now see the main Eclipse screen, consisting of (amongst others) three important regions:



   a) The package explorer: where you'll be able to manage the files containing your programmes.
   b) A large window in which you will actually write programmes.
   c) A console area in which you will see the output of your programmes.

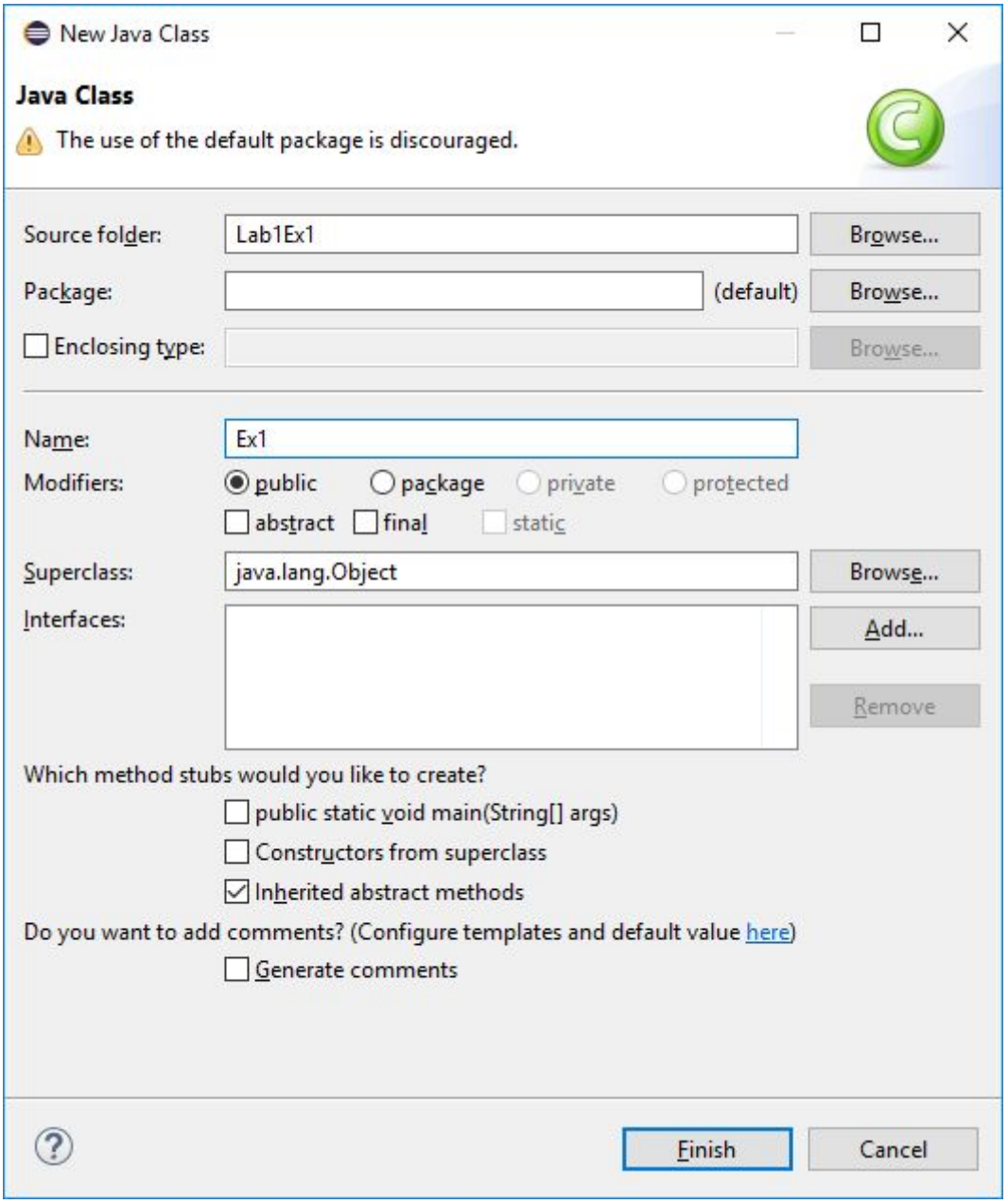
8) Time to make a programme! In the file menu, click 'New' and then 'Java Project'. Eclipse insists all Java programmes live inside a project. This is an Eclipse thing, and not something required by Java itself. You will see a window in which you can enter

properties of your new project:



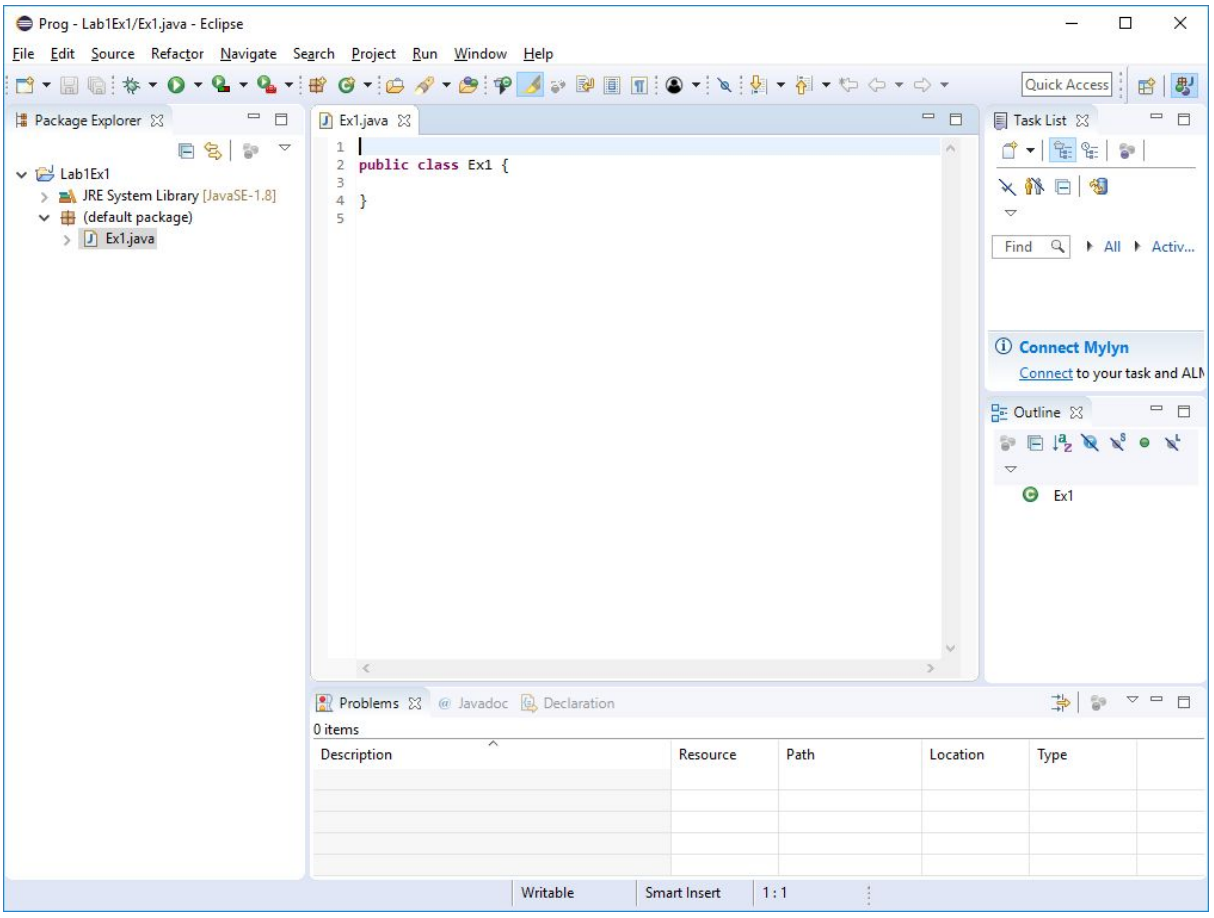
9) Give your project the name “Lab1Ex1”. Note: when programming, having spaces in project and programme names is not a good idea.
10) Click the “Use project folder as root…” option. This stops Eclipse making lots of directories which, while useful for big programs, are a bit unnecessary at the moment.
11) Click “Finish”
12) Notice that in the Package Explorer a little folder has appeared Lab1Ex1. This is an empty project - time to put a programme in it!

13) Open the File menu and click New again. This time choose “Class”. All Java programs consist of Classes. We will learn more about what Classes are in due course.

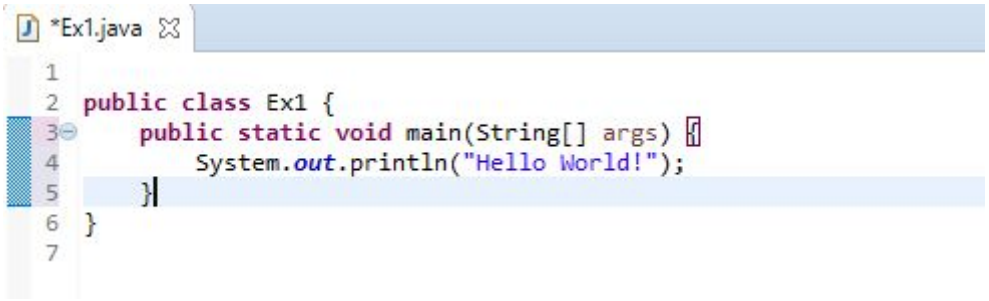


14) Give your class a name (I’ve used the name Ex1). By convention, all Java Classes have names starting with capital letters. Try and get into the habit of doing this! And remember, no spaces! Having chosen a name, click “Finish”.
15) This process will have made a file called “Ex1.java” in the “Lab1Ex1” project folder in your Eclipse workspace. Eclipse will also have automatically opened this file and you’ll see it in the large central area. Eclipse has also populated it with some lines of

program:

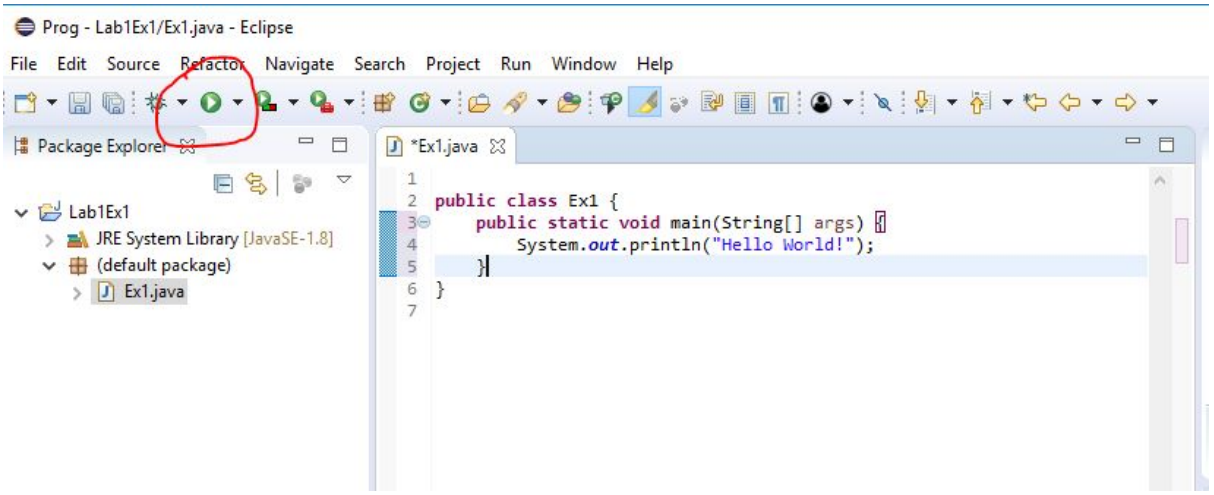


16) These lines tell Java that this file describes a class called “Ex1”. We will learn why it says “public” later.
17) After all that, you’re now ready to actually do some programming. Savour it, as it won’t take very long...add the three lines shown below to Ex1.java

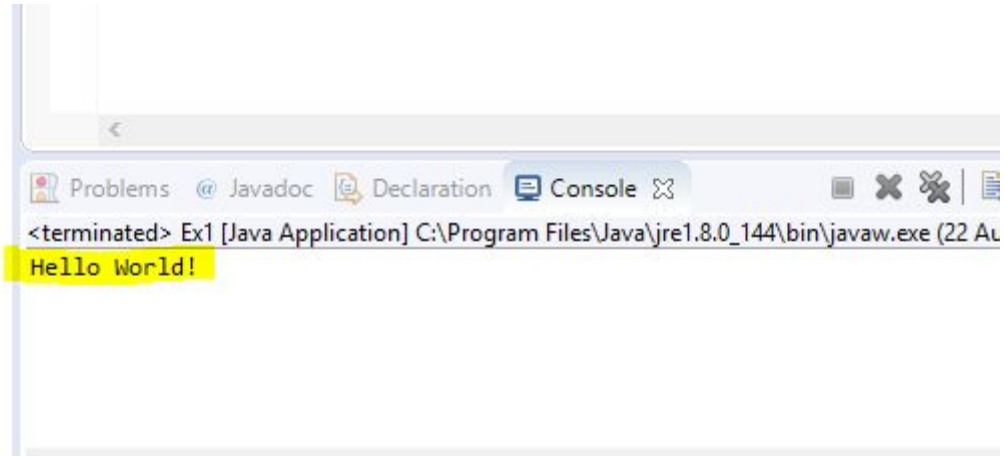


18) Note that when giving Java instructions we have to be incredibly precise. Any slight errors will cause problems. Eclipse can detect some errors and will underline things in red. Some common errors are:
    a) Forgetting capital letters (e.g. at the start of String and System)
    b) Forgetting the semicolon (‘;’) at the end of a line.

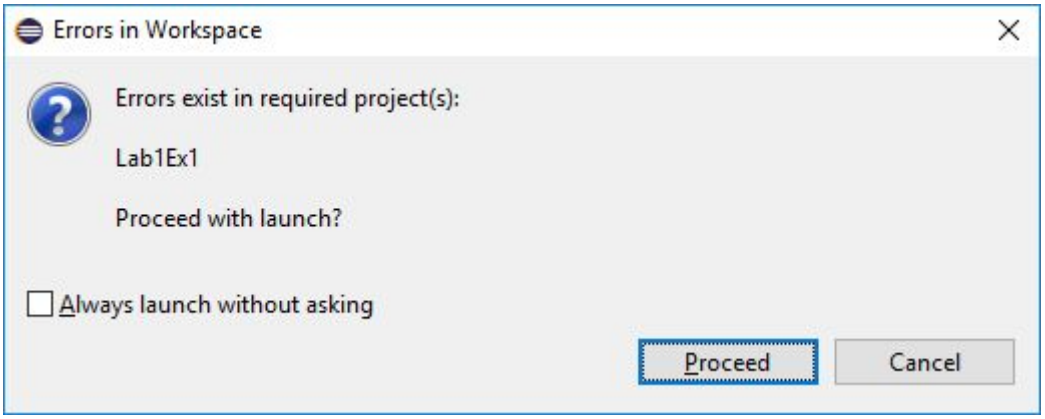
19) Once you have copied this exactly (you will understand it all in due course), it's time to run the programme. Click on the 'play' button (highlighted in red here):



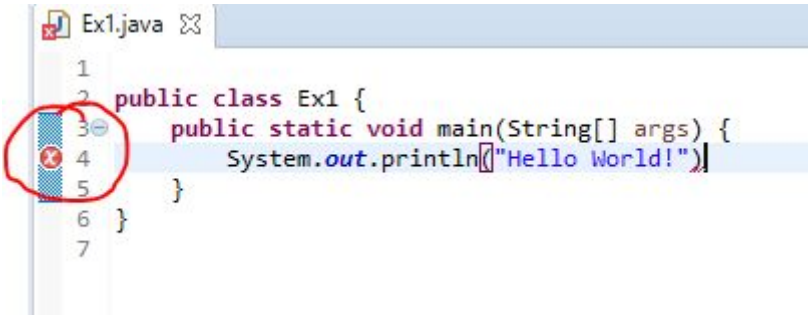
20) Eclipse will ask if you want to save your program. Click OK to save it before running.
21) The program will now run, although it's easy to miss it! Look down at the console area of Eclipse and you will see "Hello World!":



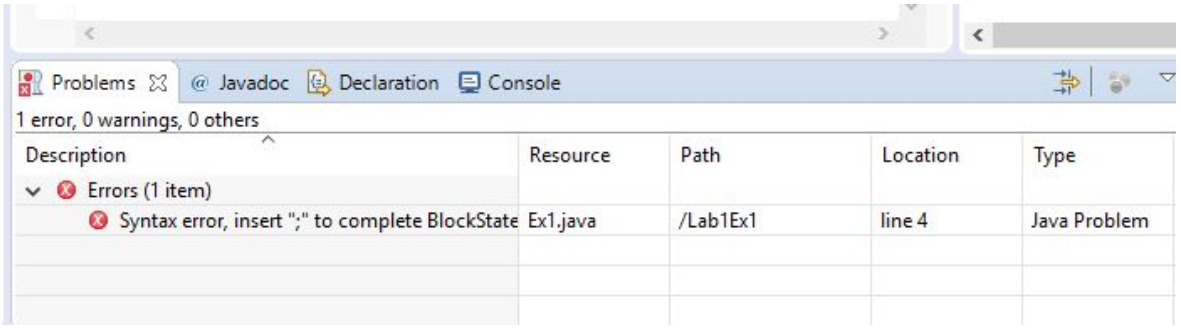
22) Let's now play around with changing the progam a bit and see what happens. Firstly, change the "System.out.println" to "System.err.println" and then run the program. What happens?
23) Now, remove the semi-colon (';') from the end of the same line and run the programme. You will now see the following error when Eclipse tries to compile:



24) This is because you have a syntax error in your programme - Jave can't understand what you want and cannot therefore compile the code. Click cancel and look back at your program. You will see that Eclipse actually detected the error (there is a red cross to the left of the offending line):



25) In the console area at the bottom of Eclipse, click on the 'Problems' tab and then expand the Error to see its details. Eclipse can detect and help you fix lots of errors like this. Here it has correctly noticed that you need to insert a semi-colon:



26) Sometimes you programs will compile happily but give errors when run. Put the semi-colon back and instead remove "[]" from beside the word "String" in the line starting "public static void" and run the program. This time Java can compile the code (it has no syntax errors) but when it tries to run it, an error will appear in the Console ("Error: Main method not found..."). We will be able to decode this error later, but for now, it's important to remember that Eclipse can pick up syntax errors, but there are other errors (known as run-time errors) that will only emerge when the program is running.
27) [optional] Although Eclipse is a very useful tool, it's informative to perform the same operations without using Eclipse (write code in a text editor and then compile and run it with commands at the command line. If you want to try this, grab a lab demonstrator and ask them to show you how!
28) If you have plenty of time, play with your program some more. Here are some example tasks to try:
    a) Can you change the text that your program displays from "Hello World!" to something else?
    b) Can you make your program display two different lines of text?

2 Variables

2.1 My first variable

In this exercise you will create your first variable. Variables are things in which we can store information and are fundamental to programming.

1. Open Eclipse, making sure your workspace is set correctly.
2. Create a new Java Project called Lab2 (remembering to select the option “Use project folder as root…”)
3. Within this project, create a new Class. Call it Lab2Ex1.
4. Copy the following code into the Lab2Ex1.java file:

```
public class Lab2Ex1 {
	public static void main(String[] args) {
		String message = "Hello World!";
		System.out.println(message);
	}
}
```

5. Before you run the program, try and predict what will happen. Run it. Was your prediction correct?
6. Now modify it to do the following and run it:

```
public class Lab2Ex1 {
	public static void main(String[] args) {
		String message = "Hello World!";
		System.out.println(message);

		message = "Goodbye!";
		System.out.println(message);
	}
}
```

7. Can you explain what is happening?
8. Try changing the line `message = "Goodbye!";` into `String message = "Goodbye!";` What happens? Why?

2.2 Types

Variables allow us to store data. Although we can change the data we store in a variable (we change what we store in message in the previous exercise), any particular variable can only store data of one type. In this example we will meet the `String` type again, as well as two types for storing numbers: `int` (integer (or whole) numbers, e.g. 1,2,3,4) and `double` (real numbers, e.g. 3.4, 8.1, 123.456)

1. Create a new class (Lab2Ex2) and enter the following code:

```
public class Lab2Ex2 {
	public static void main(String[] args) {
		String a = "Hello!";
		String b;
		int c = 5;
		double d = 3.2;
	}
}
```

2. Save and run this code. We didn't ask for any output (e.g. no System.out.println command) so we don't see anything, but the program should run just fine.
3. Note that Eclipse underlines some things in orange. These are warnings. In this case, they are warning you that you are creating variables and not using them. Hover your cursor over some code underlines in orange and read the message that Eclipse provides.
4. Play around with the things to the right of the equals (=) signs. What happens if you try and assign an integer (int) variable a value that is not an integer e.g.: `int c = "Hello!";`
5. Remove any errors you've made and add the following line and see what happens: `System.out.println(d);` Did you guess correctly?
6. Try swapping d for b in this new line. You should now get an error. Read the error and see if you can add something to your program to make it go away!
7. Sometimes you can force Java to interpret one type as another. Add the following two lines to the end of your program:

```
c = (int) d;
System.out.println(c);
```

8. Here we are telling Java to convert `d` (a `double` variable) into an `int` and storing the result in `c` (this is called casting). Note that this overwrites the value (`5`) that was previously stored in `c`. What happens to the .2 in the 3.2 when it gets interpreted as an `int`? What happens if `d = 3.6`?
9. It is not always possible to cast from one type to another. Try this example and and confirm that it doesn't work. What kind (syntax or run-time) of error does it produce?:

```
String f = (String) d;
```

10. Remove the word `int` from the line: `int c = 5;`
11. Run the program. You should get an error because you are attempting to assign a value to a variable `c` that doesn't exist. The line `int c = 5;` does two things. It **creates** a variable and **assigns** a value to it. The line `c = 5;` just **assigns** a value to a pre-existing variable. It is also possible to create a variable and not assign a value. For example: `int c;` Based on this knowledge, add code to your programme that creates a `double` variable called `pi` and assigns it the value `3.14125`. Do this twice: once using two lines, and once using just one.

2.3 Mathematical operations with variables

Once we are able to store data, we can start manipulating it. We will now do some mathematical operations with `int` and `double` variables.

1. Create a new program (a new Class) and add the following code:

```
public static void main(String[] args) {
	int a = 5;
	int b = 7;
	int c = a + b;
	System.out.println(c);
}
```

2. Run it. Is the result what you expect?
3. Try replacing the + sign with other mathematical operations: - (minus) and * (times).
4. Now, try replacing the line in which `b` is created with:

```
double b = 7.5;
```

5. Eclipse will give you an error on the following line. Can you work out how to fix it? Hint: Java doesn't mind adding together an `int` and a `double`, but the result is no longer an `int`...
6. [optional] If you really wanted to store the result of `5 + 7.5` as an `int`, what could you do?
7. Replace the line in which c is created with:

```
int c = a + 8;
```

8. Run this, it should work fine. Here the 8 is known as a literal: a hard-coded value. Can you think of an implication literals have on how you name your variables? Hint, try making a variable with the name 8!
9. Revise your code so that it looks like this:

```
public class Lab2Ex3 {
	public static void main(String[] args) {
		int a = 5;
		int b = 7;
		int c = 12;
		int d = c*(a + b);
		int e = c*a + b;
		System.out.println(d);
		System.out.println(e);
	}
}
```

10. When combining mathematical operations, we need to be careful about the order in which they will take place. Although the expressions for `d` and `e` might look the same, they are not. Java does multiplication **before** addition. So, when calculating `d` it first adds together `a` and `b` and then multiplies the result by `c`. When computing `e` it first multiplies `c` and `a` and then adds the result to `b`. To control the order, we put

things in brackets, and these are evaluated first. Add the following line and see if you can predict the result: `int f = c*(a + b*(a + b));`

11. Division is performed with the character "/". Add the following lines to your program:

```
int f = c / a;
System.out.println(f);
```

12. Is the output as you expect? We have to be careful with division in Java. When used with int variables, "/" performs an integer division. Try modifying the first of these lines to the following:

```
double f = c / a;
```

13. It is now doing an integer divide but casting the result to a double. Finally, try:

```
double f = (double) c / a;
```

14. Now `c` is being cast to a `double` prior to the division. When Java sees a `double` in a division, it does what we would think of as a standard division. The "%" character gives the remainder after integer division. E.g changing the line to the following would result in the output of 2 (12 divided by 5 is equal to 2 with remainder 2):

```
int f = c % a;
```

2.4 String operations

1. What do you think the following code would do?

```
String firstName = "Simon";
String lastName = "Rogers";
String fullName = firstName + lastName;
System.out.println(fullName);
```

2. Create a program (a Class) and try it out. Does it do what you expect?
3. This operation is called concatenation. You can concatenate any number of strings together. Add another String into the concatenation in the previous example so that the output looks neater. Hint: Simon Rogers looks neater than SimonRogers…
4. We can also use concatenation to combine Strings with other things. Try this:

```
String nonsense = "My favourite number is ";
int theAnswerToEverything = 42;
System.out.println(nonsense + theAnswerToEverything);
```

5. Try the same with a `double` variable. Some notes:
    a. Rather than store the concatenated String as another variable (e.g. the `fullName` variable above) we performed the concatenation directly in the `println` command. This saves a line, but means we cannot access the result of the concatenation later on.
    b. You can have really long variable names. This can be a good thing, as it's important to name your variables such that someone else can have a good idea of what your code does. Variable names like `a` and `b` aren't much help!
6. Write a program that starts with two integer variables `numberOfPies` and `numberOfPeople`, set to values of your choice. Perform integer division to compute

the number of whole pies each person gets, and how many remain. Store the two results in new variables (with sensible names). Print out a line that says “There are XX pies each, and YY leftover” where XX and YY should be the results of your calculations.

3 Methods and Scope

Methods

1. Write a short program that has a main method that prints a message into the console.
2. Refactor your code so that you write a method that does the printing, and your main method **calls** that method instead of doing the printing itself. Don’t worry about **passing** anything to the method, just define the text to be displayed in the method.
3. Improve your code from the previous part so that the text to be displayed is **passed** to the method when it is called by the main method.
4. (This is from the class exercise in the lecture) Write a method that computes the area of a circle when you pass it the circle’s radius (area = pi * radius * radius). Write a main method that demonstrates your method by calling it and printing out the result for two different radius values. Hint, your main method will look something like this if the two radius values are 5.6 and 1.2:

```
public static void main(String[] args) {
      double r = 5.6;
      double area = compArea(r);
      System.out.println(area);
      r = 1.2;
      area = compArea(r);
      System.out.println(area);
}
```

5. Methods within methods. Methods can call other methods. Building on the area of a circle example (above), write a second method (in the same program) that computes the circumference (call it `compCircumference`) of a circle (circumference = 2 * pi * radius). Write a third method called `summariseCircle` which is passed a radius and **computes and displays** the area (using `compArea`) and the circumference (using the circumference method). I.e. your main will call `summariseCircle` and it will call `compArea` and `compCircumference`.
6. Recall from the lectures that variables defined in methods are only visible within those methods (their scope). Write some code that demonstrates this. Define a function that takes a String as an argument and then appends something to it within the function. In your main, show that the value of the String variable passed to the method is unchanged.

4 Conditions

1. (this is from the class leap year example) Implement your code for computing if a particular year is a leap year. Test your code with a suitable set of test cases.
2. [optional] Can you make your `if` statements from the previous question more simple?
3. Refactor your code to use a method that is given a year (as an integer) and computes where it is a leap year or not. The method show return a `boolean` variable.
4. Write some code that, given a month (encoded as the first three letters of the month's name as a `String`, e.g. Jan, Feb, Mar, etc) and an integer for the year, prints out how many days there are in the month. E.g. if the month is Mar, it has 31 days, if Sep, 30 days, if Feb, 28 or 29 depending on if it is a leap year. Hint, you might want to use or (||) to test for multiple months at once (e.g. detect if it is one of the ones with 30 days)
5. Refactor again so that the code for the number of days in the month is its own method, that itself calls the method to determine if it is a leap year or not. This new method will be passed two arguments (the `String` for the month and the `int` for the year) and will return an `int` (the number of days in the month).
6. Write a program that can calculate how many whole years old someone is from today's date and their date of birth. You could store each date as three integers (day, month, year).

5 Keyboard input

We will now look at how we can get user input from the keyboard. We will gloss over some of the details here, but give you just the information you need to start getting input and processing it.

1. Copy the following code into a new Class in a new project:

```
import java.util.Scanner;
public class Lab3Ex1 {
	public static void main(String[] args) {

		Scanner keyboard = new Scanner(System.in);
		System.out.println("Please enter your name, followed by the
return key.");
		String userEntry = keyboard.nextLine();
		System.out.println("Hello " + userEntry);
	}
}
```

2. Try it out: you'll need to click in the console to type your input in. `System.in` is the Java object that collects the data coming from the keyboard. The `Scanner` provides

some help in processing that. Note that keyboard is a `Scanner` variable, in the same way that `userEntry` is a `String` variable. Troubleshooting:
    a. Don't forget the `import` line at the top. It tells Java that you want to use a `Scanner`.
3. Add the following, to capture the user's age. `Scanner` helps us here by looking for an int in the input (`nextInt()`):

```
System.out.println("Please enter your age");
int userAge = keyboard.nextInt();
keyboard.nextLine();
System.out.println("You are " + userAge + " years old");
System.out.println("What is your favourite colour?");
String col = keyboard.nextLine();
```

4. Can you remember from the lectures why we have the extra `keyboard.nextLine()` after we get the integer? If not, look back at the slides. Try removing it and see what happens!
5. Using your `summariseCircle` method from the methods exercises, write a program that asks the user to enter the radius of a circle and then, using the radius, calls the `summariseCircle` method. Your radius should be a double...fortunately `Scanner` helps us here with `nextDouble()`.
6. This exercise explores your understanding of the Scanner. Write a program that asks a user to enter their first name and their age (with a space in between, e.g. they type "Simon 123") followed by return. Use s.nextLine() to get this information as a String. Pass this String to a second scanner object (make a new one with `Scanner t = new Scanner(theString);`) and, use this to separate the two fields (t.next() will give the name and t.nextInt() will get the age). What happens if the user enters their first and second names separated by a space as well as the age (e.g. "Simon Rogers 123")? Add an extra line that fixes this problem.
7. Extend your solution from part 6 in the previous section (calculating how many whole years old someone is) to take their date of birth from keyboard input. You can decide how to ask them to enter their date of birth. E.g. you could use three different questions ("in what year were you born?", "in what month…?", etc) or get them to enter the DoB in one line separated by spaces (e.g. 3rd April 1984 would be "3 4 1984").

6 Loops

1. (class exercise 1) Write a program (all in a main method) that will, for any particular integer, print out the first ten terms of its times table.
2. If you want to practice structuring things with methods, put your times table code into a method. You should pass it the integer you want the times table for. Extend it by also passing an integer which says how many entries to show (e.g. 5 would show the first 5 terms in the times table).
3. (class exercise 2) Write a program that computes all of the prime numbers between 2 and N. This is a bit trickier than previous exercises. Look at how we broke the

problem down in class and implement and test little bits as you go along. Hint: perhaps start by writing a method than can work out if a particular integer is a prime or not.

4. I think a complete solution to part 3 needs 2 loops. What kind of loops did you use? Could you change one to a different type of loop?
5. Write a number guessing game. Your program should choose a random number between 0 and 99 (see below). The user should then be given 10 guesses to guess the number, with your program telling them, on each guess, whether they were too high, too low, or correct. To generate a random int between 0 and 99 use the following code:

```
Random rand = new Random();
int randomNum = rand.nextInt(100);
```

And add the following line to the very top of your program (above the public class line):

```
import java.util.Random;
```

6. [New!] (taken from https://www.w3resource.com/c-programming-exercises/for-loop/index.php) Write a program that, provided with the number of rows (in this case 4) produced the following triangle:

```
   1
  2  2
 3  3  3
4  4  4  4
```

I.e. you need to use the correct number of spaces to lay out the triangle. To add spaces to a string, you could write a loop that adds one space (s += " ";) each time around.

7. The double version of integer division. Write a program that for two doubles (a and b) it computes how many times b can fit into a, and the remainder. For example, if a = 4.2 and b = 1.6, then b can fit into a twice (2 x 1.6 = 3.2) with a remainder of 1.0 (4.2 minus 3.2).
8. Write a method that pads a String with a character of the users choice to make it up to a particular length. E.g. if I want to pad the string "Simon" with @ characters to make it a total of 10 characters, I would get:

```
@@@@@Simon
```

Your method should take three arguments: the String, the padding character, and the total length desired. Note that for some String `s`, you can get its length (as an integer) using: `s.length()`;

9. Paper, scissors, stone. In the game paper, scissors, stone, users both choose one of paper, scissors or stone. A set of rules then determines who wins. Write a program that allows you to play the game against the computer. You should enter your choice in the keyboard and the computer should randomly choose (earlier, I gave you the code to choose a random number between 0 and 99, you can modify it to give 0 to 2 and then use some ifs to map that to the different options). The game is repeated, and the winner is the player who is the first to win three rounds. The rules are:
    a. Paper beats stone (it wraps it up)

b. Stone beats scissors (it blunts them)
c. Scissors beat paper (they cut it up)

7 String formatting

1. In the previous exercises, you wrote a program that could show entries in a times table. Modify your code so that it works for `double` as well as `int`. E.g. your code should be able to show the times table for any double number (e.g. 8.24579). To display the results use String.format, specifying the format string for the doubles so that the layout works well even when we get a long way through the table. Assume that the number you're computing the times table for is between 0 and 99 and that the maximum term you'll go up to is 99. The output should look like:

   ~~.~~~ x ~~ = ~~~~~.~~~

   where the ~ show how wide each field should be. Write this as a method.
2. In the previous exercise we assumed various ranges for the values passed to the times table method. Add some conditions to the start of the function that check whether or not these conditions are met, and only draw the table if they are. You might find it useful to know that you can use the command `return;` to jump out of a method at any time (a bit like `break` does for loops).