Tutorial 0.2: Conditionals and Loops

1. Tutorial Objectives

By the end of this tutorial you will be able to write simple Java programs. It is expected that you have read and completed the Introduction to Eclipse tutorial already. It is important that you finish all exercises BEFORE moving on. You should also recognise that any topics that you are working on, are only the starting point for learning. You should consider finding more questions online and build up a repository of example works. As mentioned in the lectures - the best programmers are the most experienced ones. This is not about doing it once only; there are many variations on solutions and by the end of the course, your level of experience is what will determine your higher marks.

There are three levels of difficulty of the given tutorial exercises marked with asterisks (*) as follows:

* Easy   ** Medium   *** Hard

It is important to do the tutorial exercises in order as they often rely on a previous question to be completed first. Hints will be marked in italics.

You are expected to be able to finish the easy and medium exercises without any help. However, never hesitate to ask the demonstrator if you have any questions.

2. Exercises

Below exercises introduce some basic Java programming concepts. If you have never done any programming before, some parts might seem unclear, but don't let that bother you. For now, when you see things like

public static void main(String[] args) {

just copy them and and get acquainted with the structure of a program. In time you will find out what each statement means and find that it's all easier than it looks.

** Exercise 1   Javadoc*

It is important to comment all of your code so that it is readable by others and the author of any file is identifiable. Every program you write should start with a Javadoc comment as shown below.

1. Modify the HelloWorld.java from the Eclipse Tutorial to resemble Listing 1 and fill in your name and the current date. Note that the <dl> tag is a HTML notation.

```
1. /**
2. * <dl>
3. *  <dt> Purpose:
4. *  <dd> Simple example program
5. *
6. *  <dt> Description:
7. *  <dd> This program prints out the message Hello World.
8. * </dl>
9. *
10.* @author Your name
11.* @version $Date: 2015/09/18 12:29:11 $
12.*
13.*/
14.
15. public class HelloWorld {
16.   public static void main(String[ ] args) {
17.         System.out.println("Hello World");
18.   }
19. }
```

Listing 1: HelloWorld.java stub file

Hint: If you make a syntactic mistake, Eclipse will underline the problematic line in red to notify you. Always remember to run your program to confirm it works as desired.

2. Hover over the *HelloWorld* class definition on line 15 from Listing 1 and confirm that the entries from the /** documentation */ comment correctly appear in a yellow pop-up window.

* Exercise 2 Multiline Print

1. Create a new project called Induction_Tutorial _2. It is recommended to keep the tutorials separated in their own projects so that it is easier for you to look through them in the future. From now on it will be your responsibility to do so for each new tutorial.

2. To write a new program, you need to start with a new class file, otherwise you would overwrite your answer to the previous exercises. It is up to you to come up with a suitable name for each class. It helps to make them as descriptive as possible, such as ExerciseTwo in this case.

Hint: Long names are not a problem, as Eclipse provides auto-completion.

As you type a Java keyword or a name of your class, press Ctrl + Space and Eclipse will offer possible matches or fill in the rest of the word straight away.

3. Copy the program from HelloWorld.java and add another call to also print the current date. See what happens when you change *println()* calls to *print()*.
4. Modify the program so that it has the same output, but only uses one *println()* call.

Hint: Escape sequence ‘ \n’ is a newline character.

** Exercise 3 User Input*

So far we’ve seen how to make the computer display some information to us. What if now we have something to say? Let’s write a program, that will be able to “listen” to our input:

Hint: Recall from the previous tutorial that any line starting with // is a comment - it does not have any effect on what the program does, it is there solely to assist you.

```
1. public static void main(String [] args) {
2.    Scanner scanner = new Scanner(System.in);
3.    System.out.println("What is your name?");
4.
5.    // save user’s input in a String variable called ’name’
6.    String name = scanner.nextLine();
7.
8.    // print welcome message on screen including the saved input
9.    System.out.println("Hello! I thought that your name might be "
10.   + name + "!");

      }
```

Listing 2: Personalized greeting program. Note that just the *main* function is shown here - the rest, including the class declaration and comments can be taken from the previous exercises. Also note that *println* statement in line 9 is all on one line, despite of how it appears here.

Hint: Don’t forget to create a new class for each exercise to keep all your previous answers intact.

1. Don’t worry if you don’t understand everything in the given code snippet - it will become clear soon enough. For now just copy the text into your new class. You should see *Scanner* underlined with a red squiggly line - that is because your program now uses a Java library that is not included by the compiler by default. To get rid of the error go to ***Source -> Organize Imports***, which will automatically import all the libraries that we need. Try running the program and looking at the output. Try to figure out which line of the program you’ve written is responsible for what.
2. *(Optional)* Can you make the program ask your first name, then second name and then display both at the same time?

** Exercise 4   Conditionals*

So we can now communicate with our program by giving it some information. However, what if someone cheekily does not enter anything when asked for their name and just presses *Enter?* We will be left with an awkward message *I thought your name might be !*. To avoid this, let's check whether user input contains anything - we can use the conditional *if..else* statement to do just that.

```
public static void main(String [ ] args ) {
      Scanner sc = new Scanner(System.in);
      System.out.println("What is your name?");
      String name = sc.nextLine() ;

            if (name.isEmpty()) {
                  System.out.println("Fine, don't tell me your name.
                                      Goodbye!");
            } else {
                  System.out.println("Hello! I thought that your
                     name might be "+ name + "!");
            }
      }
```

Listing 3: Passive-agressive, personalized greeting program :

1. Copy and run the above program. Can you see what is happening and why?
2. (Optional) Can you change the program to react differently if you enter a name shorter than 2 characters?

Hint: The statement *name.length() < 5* will be *true* if name has less than 5 characters.

** Exercise 5   Loops*

If you wanted to print some message 5 times in a row you could just write the same code 5 times. What if you wanted to do it 500 times? Whenever you have a task that you want to perform many times, it's a good idea to use loops. Loops let you define an operation and execute it however many times you want very easily. Let's have a look at how they are defined.

```
public static void main( String [ ] args ) {
   // print the message "Hello Worlds" 10 times in a row
   for (int counter = 0; counter < 10; counter++) {
                  System.out.println("Hello Worlds");
            }
}
```

Listing 4: Example *for* loop

If you've never done any programming before, this might look a bit incomprehensible, but don't worry — in the exercises below you can just use the *for* loop like it is given above, replacing only the body (line 4). If you want to understand a bit more about how it works try searching for “java for loop” online. You will also find out more during your lectures.

1. Copy the *for* loop from the above examples and run the program. Do you see how looping can help you perform repetitive tasks quickly?

2. Replace the loop body to print

   message 0
   message 1
   message 2
   ...

   etc., where the numbers change at every iteration.

Hint: Note that the variable counter changes value at every iteration going from 0 to 9. You can append it to a string by writing

println("message " + counter).

3. (Optional) Can you make the loop run backwards from 9 to 0?

4. (Optional) You are now familiar with for loops in Java. Go online and find examples/tutorials explaining what is a *while* loop. Can you use a while loop to produce the same output as question 2?

Hint: Because *while* loops do not automatically change the value of the *counter* variable you will need to do it yourself. You can use *counter += 1*; to increment the value stored in the *counter* variable by 1.

~~~ 000~~~
~~~