CSCU9P1 - Programming Practical 2: Java practice

AIMS

- To practice writing small programs using numbers and strings.
- To learn more about syntax errors.
- To learn how to create a very simple Graphical User Interface.

If you get stuck or need help, ask a demonstrator for help during a practical session.

This sheet contains one formative milestone and one class test preview milestone. It is **strongly** recommended that you should show and discuss your class test preview milestone to a demonstrator.

Part One: Java Programming Exercises

You are asked to write Java programs to perform a number of tasks. For each task, you must create a separate BlueJ project containing the Java program for that task. Refer to the last practical sheet if you need a reminder of the steps involved in creating, compiling, and running a Java program in BlueJ.

There is somewhat less guidance provided in this worksheet compared to the last worksheet. We expect you to think creatively, refer to the lecture notes and textbook, and work out your own solutions. However, do not allow yourself to remain stuck: ask for help if you need it.

Task 1. Greeting

Write a program that prints a friendly greeting of your choice in a language other than English.

[**Hint**: instead of creating a new BlueJ project from scratch, start with your **HelloWorld** project from the last practical. From BlueJ's **Project** menu, use **Save As** to save a copy of this project with a name of your choice. Then modify the program code, remembering to change the name of the main class from **HelloWorld** to something more appropriate. Notice that when you change the name of the class, BlueJ automatically changes the name of the file containing the code to match the name of the class.]

Task 2. Simple Arithmetic

Write a program that prints a line on the screen as described below.

First, use the command `System.out.print() to` print the words:

```
The sum of the first ten positive integers is
```

on the screen without moving to a new line. Include a space after the final word. Then, on the same line, print the sum of the first ten positive integers, 1 + 2 + ... + 10. The program must calculate this sum and print the resulting value on the screen.

Task 3. Syntax Error Bingo

This task is a little different. Start with a copy of any BlueJ project you have created; it does not matter which. Make sure you are working with a *copy*, so that you don't overwrite the original project. Give the copied project an appropriate name, *e.g.*, **SyntaxErrorBingo**.

Your task is to introduce syntax errors into the program to get the compiler to show the error messages listed below. You do not have to get all of the error messages at the same time. Take note of the change you made to produce each error and try to work out exactly what has gone wrong. There are some rows left blank in case you come up with error messages that are not in the list.

If you need help with any of the error messages, ask on the Canvas discussion boards, on MS Teams chat, or at your next meeting with your tutor.

Error message	**Change that produced this error message**
; expected	
unclosed string literal	
')' expected	
reached end of file while parsing	
cannot find symbol...	
class, interface or enum expected	
not a statement	

Formative Milestone Intro B1 Difficulty: ✷ Time: ✷

To reach this milestone, you must complete the three tasks in Part One.

You do not have to show this work to an instructor unless you want to get feedback or help. Use the Intro Formative Milestone B1 quiz on Canvas to evaluate your work.

Part Two: Fun with GUIs

Graphical User Interfaces, or GUIs, are part of most modern software. GUI features include windows, pop-up dialog boxes, drop-down menus, clickable icons, mouse input and many other familiar concepts. In semester 2 you will learn to write Java programs using a full range of GUI features. This semester we will focus on the core constructs of Java, but in this part of the practical we take a sneaky look ahead at a simple GUI construct and use it to have some fun.

- Create a new BlueJ project called **HelloWorldGUI**. Add a new class to the project containing the following code:

```
import javax.swing.JOptionPane;

public class HelloWorldGUI

{

  public static void main(String[] args)

  {

    JOptionPane.showMessageDialog(null, "Hello World!");

  }

}
```

Compile and run this program in the usual way.

Take a look at the first line in this program. It is an example of an `import` statement. One of the biggest strengths of Java is that it is supplied with a vast array of *library packages* that provide many useful and diverse functions. The `import` statement is used to import a library package into a program so that the functions it provides can be used. Your program is importing part of the Swing library package, which provides a large range of GUI constructs.

The `main()` method in this program is very similar to the `main()` method in the **HelloWorld** program you wrote in the last practical. The difference is in where the output appears. The **HelloWorld** program used `System.out.println()` to write "Hello World!" on the terminal window (usually called *standard output*). This program, however, uses a method from the Swing library to create a GUI object (a `JOptionPane`) in which "Hello World!" is displayed. The `JOptionPane` remains on the screen until the user clicks "Ok".

Let's play around with some more programs using `JOptionPane`s.

- Your next task is easy. Save a copy of your **HelloWorldGUI** project with the name **HelloMeGUI**. Modify the code in the new project so that the program shows "Hello " followed by your name. A `JOptionPane` can also be used to get information from a user. The program can store this information in a *variable* and use it later on. (You will learn more about variables in lecture 2.)

Your next program will carry out some simple interaction with the user.

- Create a new project called, say **HelloUserGUI**. (As before, the easiest way to do this is to use **Save As** to copy one of your previous projects giving it the new name.) Edit the code so that it looks like this:

```
import javax.swing.JOptionPane;

public class HelloUserGUI

{

  public static void main(String[] args)

  {

    String name = JOptionPane.showInputDialog("What is your name?");

    JOptionPane.showMessageDialog(null, "Hello " + name + "!");

  }

}
```

Compile and run this program.

When you have seen what the program does, take a closer look at the two lines of code inside its `main()` method.

The first line is:

```
String name = JOptionPane.showInputDialog("What is your name?");
```

This uses `JOptionPane.showInputDialog("What is your name?")` to pop up a `JOptionPane` containing the prompt "What is your name?" and an area for the user to type in a reply. The user's reply is then stored in a variable called `name`.

The second line is:

```
JOptionPane.showMessageDialog(null, "Hello " + name + "!");
```

This line creates a second `JOptionPane`, this time with no user input area. This `JOptionPane` displays the text `"Hello "`, then the contents of the variable `name`, then an exclamation mark. Here, the symbol + is used to glue or concatenate two *strings* (pieces of text).

(Recall that Java also uses + for adding numbers. Having different meanings for the same symbol is called *overloading*, and is common in programming languages. The compiler is able to figure out which meaning is intended by looking at the context in which the symbol appears. In this program, can you see how the compiler can tell that the + means concatenation rather than addition?)

- For your final task, and something that is likely to appear in your class test, I want you to think of a "Knock, knock" joke. If you don't know one find one on the web. Create a program which asks the user's name, greets the user, and then delivers your joke. Use `JOptionPane`s to manage the dialogue between the program and the user. Assume that the user will cooperate and type in the correct responses, so there is no need to check what the user has typed. The interaction might proceed like this:

```
Program: What is your name?

User: Bob

Program: Hello, Bob!  Here is a joke.  Knock, knock!

User: Who's there?

Program: Cheese!

User: Cheese who?

Program: Cheese a jolly good fellow!
```

(My joke isn't very funny. Please find a better one to use!)

Class Test Preview Milestone: Knock Knock

Difficulty: ✵ Time: ✵✵

This is your class test preview milestone – there will likely be a question on the class test that looks a lot like this, but not exactly. You will be able to take this code into your test and adapt it to answer the question based on *your understanding*. Show this to a demonstrator and they will ask you a couple of questions that will help you prepare for the test.

You have now completed the milestones on this sheet but read on for some useful information about how to create a standalone version of your program that you can run without using BlueJ and conveniently share with others.

How to Share Your Programs

Are you proud of a program you have written? Would you like to share it with friends? BlueJ can be used to create a standalone version of your program that you can distribute. The recipient will be able to run the program by simply double-clicking on it. (The recipient's computer must have Java installed, but that is pretty standard nowadays.)

To share your program it needs to be turned into an executable JAR file. A JAR file (Java ARchive file) is basically a single file which aggregates and compresses all the files that make up a Java program. It is very similar to a ZIP file.

A JAR file can be made executable by specifying which file contains the program's main method.

To create an executable JAR file in Blue follow these steps:

- From BlueJ's **Project** menu, choose **Create Jar File**.

- In the dropdown menu labelled **Main class**, choose the file that contains the program's main method.
- If you want to share your source code, tick the checkbox labelled **Include source**. Leave this box unticked if you want to share the runnable program but not its source code.
- If you want to include the various housekeeping files that BlueJ creates for its own use, tick the checkbox labelled **Include BlueJ project files**.
- Click **Continue**. A dialog appears allowing you to choose a location and name for the JAR file (this does not have to be the same as the name of the project). Choose these, then click **Create**.
- That's it! The program can now be run by simply double-clicking on the JAR file. The JAR file can be distributed by email, copying, uploading to a website, or any other means.

Caveat: if your program uses standard output (e.g., if it contains `System.out.println()` statements) the JAR file will not run correctly when it is double clicked (because Java will not know where standard output is). In this case the JAR file can be run within the Command Prompt (as you did in part one of the last practical), using the command **java –jar** (filename.jar). The Command Prompt window will serve as standard output.