Lab 1: Introduction

Let's get started!

It's time to have some fun.

1. **Open Eclipse**. It's a good idea to keep all your projects in the same workspace. Think of a workspace as a shelf, and each project is a binder of stuff on the shelf. Don't store or unzip anything not created by Eclipse in this workspace.
2. **Create a new Java project**. Call it Comp1510Lab01*LastNameFirstInitial*. For example, if your name were Fred Bloggs, you would name the project **Comp1510Lab01BloggsF**.
3. **Complete the following tasks**. Remember you can right-click the src file in your lab 1 project to quickly create a new Java class.
4. When you have completed the exercises, **show them to your lab instructor**. Be prepared to answer some questions about your code and the choices you made.
5. Although you can cut and paste from the lab document, it will help you learn if you **type in the code by hand**. The tactile effort of typing in the code will help reinforce the Java syntax.

What will you DO in this lab?

In this lab, you will practice:

1. Creating a class that contains a main method
2. Printing information to the screen
3. Commenting your code
4. Recognizing and fixing syntax errors
5. Using the addition and concatenation operator.

Table of Contents

Let's get started! **1**

What will you DO in this lab? **1**

1. Poem **2**

2. Comments **2**

3. Program Names **3**

4. Recognizing Syntax Errors **4**

5. Correcting Syntax Errors **5**

6. Two Meanings of + **5**

7. You're done! Show your lab instructor your work. **7**

1. Poem

1. Create a new class called **Poem** inside package ca.bcit.comp1510.lab01
    a. Ask Eclipse to include a main method inside the class definition
2. Print the message "Roses are red." Remember the statement that prints the message must be inside main.
3. Compile and run your program. When it works correctly, modify your program so it prints the entire poem:

    Roses are red
    Violets are blue
    Sugar is sweet
    But I have commitment issues
    So I'd rather just be friends
    At this point in our relationship.

2. Comments

1. Let's create a program that counts from 1 to 5 in English, French, and Spanish:
2. Create a new class called **Count** inside package ca.bcit.comp1510.lab01
    a. Include comments
    b. Include a main method inside the class definition
3. Add the following three lines to the main method:
    a. `System.out.println ("one two three four five");`
    b. `System.out.println ("un deux trois quatre cinq");`
    c. `System.out.println ("uno dos tres cuatro cinco");`
4. Add/complete the Javadoc comment to the top of the class. The Javadoc comment should contain three things:
    a. A short description of the program
    b. The @author tag and the author's name
    c. The @version tag and the version number.
5. Add a comment before each print statement that describes what it does. Use the // style comment. Insert a blank line before each comment (in your code, not the output). Does this make your code easier to read?
6. Remove one of the slashes from one of your comment lines and recompile the program, so one of the comments starts with a single /. What error do you get? Put the slash back in.
7. Try putting a comment within a comment, so that a // appears after the initial // on a comment line. Does this cause problems?
8. Consult the documentation guidelines in Appendix F of the text. Have you violated any of them? List two things that you could imagine yourself or someone else doing in commenting this program that these guidelines discourage.

3. Program Names

A Java identifier may contain any combination of letters, digits, the underscore character, and the dollar sign, but cannot begin with a digit.

Furthermore, by convention, identifiers that represent class names (which includes program names) begin with a capital letter. This means that you won't get an error if your program name doesn't start with a capital letter (as long as what it starts with is legal for an identifier), but it's better style if you do.

This may seem arbitrary now, but later when you have lots of identifiers in your programs you'll see the benefit. Of course, the program name should always be reasonably descriptive of the program.

Suppose a file called Simple.java contains the following program.

```
public class Simple {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("I love Java");
    }
}
```

Indicate whether each name below is a legal identifier, and if so, whether it is a good choice to name our program. If the answer to either question is no, explain why. Write your answers in a comment block in the code file for the class `Simple`. Place the comment block at the end of the file, after the rest of the code.

To check your answers, try to create a new class in Java with the specified name. Does the New Java Class dialog box provide any warnings or complaints (look at the top of it):

1. simple (why is this different from the first)
2. SimpleProgram
3. 1 Simple
4. _Simple_
5. *Simple*
6. $123_45
7. Simple!

4. Recognizing Syntax Errors

When you make syntax errors in your program the compiler generates error messages and does not create the bytecode file.

You can save some time and frustration to learn what some of these messages are and what they mean. In the following exercise you will introduce a few typical errors into a simple program and examine the error messages.

1. Create a new class called **Hello** inside package ca.bcit.comp1510.lab01. It should have a comment and a main method. Modify the program so that it looks like this:

```
package ca.bcit.comp1510.lab01;

/**
 * Prints a Hello World message.
 *
 * @author BCIT
 * @version 2022
 */
public class Hello {

    /**
     * Prints the greeting.
     *
     * @param args
     *             unused
     */
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

2. Compile and execute the program to see what it does. Then make the following changes and write your answers to the following questions in a comment block in the code for the `Hello` class. Put the comment block at the end of the file, after the rest of the code.
    a. **Class name different from file name**. Delete one l (el) from the name of the class (so the first non-comment line is public class Helo), save the program, and recompile it. What was the error message?
    b. **Misspelling inside string**. Correct the mistake above, then delete one l from the Hello in the message to be printed (inside the quotation marks). Save the program and recompile it. There is no error message—why not? Now run the program. What has changed?
    c. **No ending quotation mark in a string literal**. Correct the spelling in the string, then delete the ending quotation mark enclosing the string Hello, World!. Save the program and recompile it. What error message(s) do you get?

d. **No beginning quotation mark in a string literal**. Put the ending quotation mark back, then take out the beginning one. Save and recompile. How many errors this time? Lots, even though there is really only one error. **Pro-tip: When you get lots of errors always concentrate on finding the first one liste**d. Often fixing that first error will fix the rest.
e. **No semicolon after a statement**. Fix the last error (put the quotation mark back). Now remove the semicolon at the end of the line that prints the message. Save the program and recompile it. What error message(s) do you get?

5. Correcting Syntax Errors

1. Create a new class called **Problems** inside package ca.bcit.comp1510.lab01. Replace the contents of the source file Eclipse made with the following:

```
public class problems
{
    public Static main (string[] args)
    {
        System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
        System.out.println ("This program used to have lots ofproblems,");
        System.out.println ("but if it prints this, you fixed them all.")
        System.out.println (" *** Hurray! ***");
        System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
    }
}
```

2. This creates some syntax errors. Can you fix them all?
    a. **Hint 1: Java is case sensitive**. The identifiers public, Public, and PUBLIC are all considered different. For reserved words such as public and void and previously defined identifiers such as String and System, we *must* use the correct case.
    b. **Hint 2: When you get lots of errors always concentrate on finding the first one liste**d. Often fixing that first error will fix the rest.
    c. Read the error messages carefully and note what **line numbers** they refer to. Often the messages are helpful, but even when they aren't, the line numbers usually are.
    d. When the program compiles cleanly, run it.

6. Two Meanings of +

In Java, the symbol + can be used to add numbers or to concatenate strings. This exercise illustrates both uses.

When using a string literal (a sequence of characters enclosed in double quotation marks) in Java the complete string must fit on one line. The following is NOT legal (it would result in a compile-time error):

```
System.out.println ("It is NOT okay to go to the next line
```

```
       in a LONG string!");
```

The solution is to break the long string up into two shorter strings that are joined using the concatenation operator (which is the + symbol). This is discussed in the text. So the following would be legal:

```
      System.out.println ("It is okay to break a long string into parts" +
            " and join them with a concatenation symbol.");
```

So, remember:

- When working with strings the + symbol means to concatenate the strings (join them)
- When working with numbers the + means what it has always meant—add!

1. Create a new class called **Plus** inside package ca.bcit.comp1510.lab01. Modify the program so that it looks like this:

```
package ca.bcit.comp1510.lab01;

/**
 * Demonstrates the different behaviours of the + operator.
 *
 * @author BCIT
 * @version 2021
 */
public class Plus {

    /**
     * Drives the program.
     * @param args unused
     */
    public static void main(String[] args) {
        System.out.println("This is a long string that is the" +
                " concatenation of two shorter strings.");
        System.out.println("The first computer was invented about " + 70 +
                " years ago");
        System.out.println ("8 plus 5 is " + 8 + 5);
        System.out.println ("8 plus 5 is " + (8 + 5)) ;
        System.out.println (8 + 5 + " equals 8 plus 5.");
    }
}
```

2. Compile and execute the program. Compare the output of the final three statements. Be prepared to explain what happened to your lab instructor.
3. Create a new class called **Birds** inside package ca.bcit.comp1510.lab01.
    a. Print out the statement “Ten robins plus 13 canaries is 23 birds.”
    b. You must use only one println statement
    c. You must use the + operator to perform arithmetic
    d. You must use the + operator to perform string concatenation.

7. You're done! Show your lab instructor your work.

Notice that we pay attention to style in addition to correctness. If you want to be appreciated as a professional software developer, it is critical to develop good habits that lead to clear and well-organized code. Once you get into the habit of creating readable code it becomes quite natural and effortless.

Simple things you should develop from the start include

proper indentation when coding (such as shown in the PlusText example)

keeping code lines within the 80-column limit, avoiding long lines.

Checkstyle will help you learn some of these simple habits. Fix style errors as they occur and you will make fewer errors, as you build better habits.

If your instructor wants you to submit your work in the learning hub, export it into a Zip file in the following manner:

1. Right click the project in the Package Explorer window and select export…
2. In the export window that opens, under the General Folder, select Archive File and click Next
3. In the next window, your project should be selected. If not click it.
4. Click *Browse* after the "to archive file" box, enter in the name of a zip file (the same as your project name above with a zip extension, such as Comp1510Lab01BloggsF.zip if your name is Fred Bloggs) and select a folder to save it. Save should take you back to the archive file wizard with the full path to the save file filled in. Then click Finish to actually save it.
5. Submit the resulting export file as the instructor tells you.