

Langara College

Department of
Computing Science & Information Systems

Laboratory Guide
CPSC 1150

Spring, 2024

Nothing is particularly hard if you divide it into small jobs. (Henry Ford: 1863 – 1947)

Preface

The main purpose of lab experiments is to implement those aspects of computing and programming which are discussed in class but best learned through hands-on experiments and under instructors' supervision. Through the labs you learn how to manage your time, how to write reliable, efficient, and robust programs, how to test and debug programs, and how to refine and improve the algorithms.

The assignments in this course are programming experiments based on Top-Down design and procedural abstraction. The lab/programming assignments are designed to extend your knowledge in top-down design so that you can write relatively sophisticated programs.

A designated lab assistant will assist the instructor during the lab hours and provide support to you in the open lab time. S/he will help you in getting familiar with the computing environment. This includes hardware, system software and lab facilities. Lab assistants are not supposed to debug programs or write programs for the students.

Most of the programming assignments are not designed for a one- or two-hour lab session. You should learn to budget and manage your time so that you can submit your completed work on time.

1. General Instructions

Programming assignments generally involve reading the related course materials and the assignment specification in advance. You should have the written pseudocode algorithm of your program before coming to the lab. The lab period is intended to provide an opportunity to compile, debug, and run assigned programs.

2. Equipments

2.1 Using Langara lab computing device:

Our computer labs are attached to various network file servers that have all the software you will need.

You are given a computer account which should be accessible from all computer labs on campus. You have access to shared laser printing facilities on the network via a card that can be purchased from the library.

2.2 Using your own computing device:

You need Java compiler, and simple text editor or simple IDE on your computing device to do your labs and assignments.

- Java Compiler (please refer to Appendix B of this document).
- Text Editor or IDE (please refer to Appendix C of this document).

3. Method of Submission

You must submit your completed lab/assignment work to *BrightSpace on the due dates*. A sample program can be found in *appendix A* of this Lab Guide.

NOTE

If you use ideas or code other than your own they should be acknowledged as such in any work you hand in. Plagiarism is an unacceptable conduct and is dealt with severely.

4. Documentation of Programs

Documentation refers to adding explanation notes to help other people understand your program. There are two kinds of documentation: external and internal. **External** documentation is addressed to the user and is concerned with **what** the program does, the algorithms used, and **how** to use the program. **Internal** documentation is addressed to the people who maintain the program and is concerned with the details of how the program was written and how the program works.

4.1 External Documentation

Your external documentation should include:

- the program's name
- the purpose of the program
- a list of other programs/packages, if any, that must be combined with this program, to make a workable combination
- the input layout - the correct way to type the input data (show an example)
- the output layout (output produced)
- a list of bugs you haven't fixed
- the name of any program you borrowed ideas from
- the **pseudocode** of how program works

4.2 Internal Documentation

An explanation of how the program works. Internal documentation helps other programmers borrow your ideas, and helps them expand your program to meet new situations (maintenance).

It should include:

- the program's name
- your name and your studentId
- the date you finished it
- the purpose of the program (if necessary)
- the meaning of each variable (if necessary)
- the purpose of each class as well as specifications for each method including
 1. precondition which is a statement on the conditions that must exist on the input parameters when the method is called
 2. postcondition which is a statement of the conditions after the method is executed

5. Implementation

The final step of problem solving is to implement the solution in a High Level Language such as Java. That is, convert the pseudocode into Java statements by carefully following the syntax of the prescribed language.

For example, any line beginning with a `//` is a comment to ensure that the purpose of the coded statements is understood by the programmer and any other programmers who may have to review or modify the code. Also, Java statements typically end with a semicolon (`;`) which is a statement terminator. We will learn more Java syntax as the course goes along.

6. Development Environments

For all your programming assignments you will be using the Java Development Kit (JDK), to compile and execute your Java programs. Please refer to **Appendix B** for an introduction to JDK and to **Appendix C** for an introduction to **SciTE**.

Please **note** that if you use **SciTE**, there are some menu items which directly call the JDK compiler or the JDK command to run (execute) your Java program.

6.1 Integrated Development Environments (IDEs)

Integrated Development Environments (IDEs) are something that we normally desire to use for writing, compiling, and running programs written in Java (or many other languages). They include an editor to enter and revise the source code (the program written by you), a Java compiler that translates the source code into binary machine instructions to be used by the computer hardware, and often a linker that puts together the binary code produced by the compiler from your source code with any other binary code that is needed to run the program.

All these aspects are found together in a single piece of software, so that you do not have to flip from one place to another to assemble the parts needed for a complete program. There are many of these available, ranging from complex and expensive down to reasonably simple and free!

Then there are free IDEs like **IntelliJ**, **NetBeans** and **Eclipse**. These involve some effort to learn to use, but many people like them after they have survived the learning curve.

Finally there are stripped down Java IDEs such as **SciTE**. Refer to **Appendix C** for an introduction to **SciTE**.

7. Key points to remember

- Java programs are built with Classes and methods.
- Each Java **program** has a method, **public static void main(String[] args)**.
- Program execution starts with the first statement of method `main(String[] args)`.
- Case is significant in Java. All keywords are in lowercase; user-defined identifiers may be in upper- or lower-case.

7.1 Good Programming Practices / Style Tips

- 1) Every program should begin with a comment describing the purpose of the program.
- 2) When you define classes and methods, you should thoroughly comment their behavior.
- 3) The purpose of each variable should be stated.
- 4) The program code and comments should be arranged so that the program is easy to read.
- 5) Use blank lines, space characters and tab characters in a program to enhance program readability.
- 6) By convention, you should always begin a class name with a capital first letter. When reading a Java program, look for identifiers that start with capital first letters. These normally represent Java classes.
- 7) Whenever you type an opening left brace, {, in your program, immediately type the closing right brace, }, then reposition the cursor between the braces to begin typing the body. This helps prevent missing braces.
- 8) Make sure that the braces { } are aligned.
- 9) Proper and consistent indentation should be used. Indent the entire body of each class definition one “level” of indentation between the left brace, and the right brace, that defines the body of the class. This emphasizes the structure of the class definition and helps make the class definition easier to read.
- 10) Set convention for the indent size you prefer and then uniformly apply that convention. The Tab key may be used to create indents.
We recommend using 3 to 5 spaces to form a level of indent.
- 11) Indent the entire body of each method definition one “level” of indentation between the left brace, and the right brace, that define the body of the method. This makes the structure of the method stand out and helps make the method definition easier to read.
- 12) Place a space after each comma in an argument list (,) to make programs more readable.
- 13) Choosing meaningful variable names helps a program to be “self-documenting” (i.e., it becomes easier to understand a program simply by reading it rather than having to read manuals or use excessive comments).
- 14) By convention, variable name identifiers begin with a lower case first letter. As with class names every word in the name after the first word should begin with

a capital first letter. For example, identifier `firstNumber` has a capital N on its second word `Number`.

- 15) Some programmers prefer to declare each variable on a separate line. This format allows for easy insertion of a descriptive comment next to each declaration.
- 16) Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.
- 17) Using parentheses for complex arithmetic expressions even when the parentheses are not necessary, can make the arithmetic expression easier to read.
- 18) Indent the statement in the body of an if structure to make the body of the structure stand out and to enhance program readability.
- 19) Place only one statement per line in a program. This enhances program readability.
- 20) A lengthy statement may be spread over several lines. If a single statement must be split across lines, choose breaking points that make sense such as after a comma in a comma separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines.
- 21) Refer to the operator precedence chart when writing expressions containing many operators. Confirm that the operators in the expression are performed in the order you expect. If you are uncertain about the order of evaluation in a complex expression, use parentheses to force the order, exactly as you would do in algebraic expression. Be sure to observe that some operators, such as assignment (`=`), associate right to left rather than left to right.
- 22) Special values such as interest rate, tax, the math value 3.14 (pi), etc., should be defined as named constants which are generally made of upper case letters. Never use magic numbers in the statements.
For example use **`principle * INTEREST_RATE`** instead of **`principle * 0.7`**.
- 23) Any assumption made by the program should be stated. For example, “the program assumes that

8. Assignments / Labs

You are given a weekly Lab and an Assignment to work on.

8.1 Labs:

- Labs are simple programs that should be completed in 2 hours during the Lab sessions.
- There is no need to comment your Lab programs.
- Lab programs must be written following Good programming Styles.
- Labs should be submitted to Brightspace before the end of Lab sessions. Up to 10% will be deducted from late labs (1% deduction per hour).

8.2 Assignments:

- Everyone will be required to hand-in an attempt at solving every assignment in order to pass the course. In other words, each programming assignment must be attempted and submitted (even if not completed).
- Students should submit their programming assignments on or before the announced due time and date. Up to 20% will be deducted from late assignment (1% deduction per hour up to 20%). Late Assignments can only be submitted to BrightSpace up to 48 hours after the due date.
- The Marking Scheme for each assignment is specified at the end of the assignments. In general assignments will be marked based on the following criteria:

Presentation: Organization, legibility, and readability

Design: Data types, abstraction, and design

Style: Headers, comments, descriptive identifiers, and indentation

Correctness: Program meets specifications – works without bugs

Documentation: External and internal

Testing: Program is tested using suitable and comprehensive input data

Appendix A of this Lab Guide contains a sample program with full documentation.

Please pay attention to its design, style, and documentation

Appendix A: (A sample program)

File AccountTest.java:

```

/**
** Program Name: AccountTest
** Author:      Student's Name
** Date:       May 7th, 2020
** Course:    CPSC 1150
** Compiler:   JDK 1.8
*/

public class AccountTest
{
    /**
    ** This program displays a student's information and the course information.
    **
    */
    public static void main(String[] args)
    {
        String name = "Jim Smith",
            logInName = "jsmith00",
            instructorName = "Bill Gates";
        int studentNumber = 100000000;

        showCourseInfo( instructorName );    //display the course information

        // display the student's name
        System.out.println("My name is " + name + ".");

        //display the student number and login name
        System.out.println("My student number is: " + studentNumber);
        System.out.println("My net work log in name is: " + logInName);

        System.out.println("\t*****End of Sample Code*****");
    }

    /**
    ** Displays the course information
    ** @param inputName a string stands for the instructor's name.
    ** @return none.
    ** precondition: inputName is declared in the calling function.
    ** postcondition: all the information will be displays on the screen.
    */

    public static void showCourseInfo(String inputName)
    {
        //display the course name and number.
        System.out.println("This course is CPSC 1150-003.");

        //display the instructor's name.
        System.out.println("The instructor is : " + inputName + ".");

        System.out.println("Have fun in this course.\n");

        return; //return to where this function is called (note for void function return is optional).
    }
}

```

Sample External Documentation

Program AccountTest

<u>File Name:</u>	AccountTest.java
<u>Purpose</u>	To test the Account class and its methods.
<u>Input</u>	No input is required for this program.
<u>Output</u>	Student information displays on the computer screen

Program Logic (Pseudocode)

Algorithm AccountTest

START

1. Step1

2. .

3. .

4. .

END AccountTest.

Appendix B: Introduction to Java Development Kit

The *Java Development Kit* (JDK) is a key platform component for building java applications.

The JDK is one of three core technology packages used in Java programming, along with the JVM (Java Virtual Machine) and the JRE (Java Runtime Environment). It's important to differentiate between these three technologies, as well as understanding how they're connected:

- The JVM is the Java platform component that executes programs.
- The JRE is the on-disk part of Java that creates the JVM.
- The JDK allows developers to create Java programs that can be executed and run by the JVM and JRE.

Creating Your First Application

Your first program, HelloWorldApp, will simply display the greeting "Hello world!". To create this program, you will:

- **Create a source file.** A source file contains text, written in the Java programming language, that you and other programmers can understand.
- **Compile the source file into a bytecode file.** The *compiler*, `javac`, takes your source file and translates its text into instructions that the *Java Virtual Machine* (Java VM) can understand. The compiler converts these instructions into a bytecode file.
- **Run the program contained in the bytecode file.** The Java interpreter installed on your computer implements the Java VM. This interpreter takes your bytecode file and carries out the instructions by translating them into instructions that your computer can understand.

`javac HelloWorld.java`

Java HelloWorld

a. Create a Source File.

Start a simple text editor like Notepad.

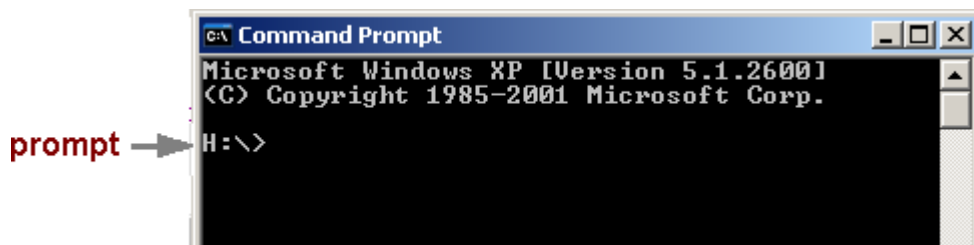
In a new document, type in the following code:

```
/**
 * The HelloWorldApp class implements an application that
 * displays "Hello World!" to the standard output.
 */
public class HelloWorldApp {
    public static void main(String[] args) {
        // Display "Hello World!"
        System.out.println("Hello World!");
    }
}
```

Save this code to a file named **HelloWorldApp.java**

b. Compile the Source File.

Select the Command Prompt application. When the application launches, it should look like this:



The prompt shows your *current directory*. To compile your source code file, change your current directory to the directory where your file is located. For example, if your source directory is CPSC1150\lab0 on the H drive, you would type the following command at the prompt and press Enter:

```
cd H:\CPSC1150\lab0
```

Now the prompt should change to H:\cpse\1150\lab0>.

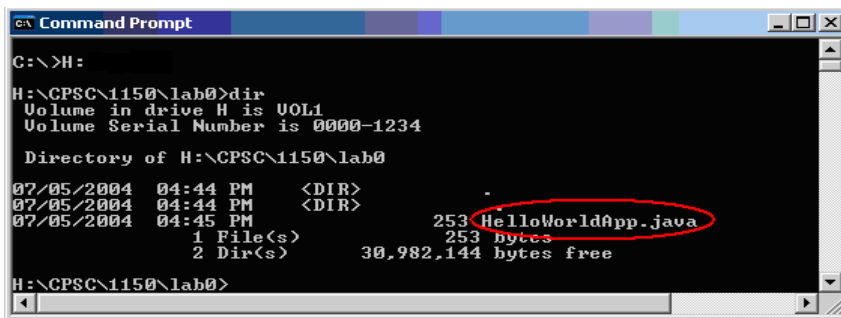
Note: To change to a directory on a different drive, you must type an extra command, for example to change the current drive to C drive, type:

C:



As shown here, to change to the CPSC1150 directory on the H drive, you must reenter the drive, H:

If you enter **dir** command at the prompt, you should see your file.



Now you can compile. At the prompt, type the following command and press Enter:

```
javac HelloWorldApp.java
```

If your prompt reappears without error messages, congratulations. You have successfully compiled your program.

Java Environment Variables:

The tools in the Java Developers Kit (JDK) use the following environment variables:

- PATH – The normal executable search list should include ... \jdk1.7.0_40\bin
- CLASSPATH – A colon-separated (:) list of directories containing compiled .class files for use with applications and applets

C. Run the program

After compiling your program successfully the HelloWorldApp.class file will be created.

To run your program Enter:

```
java HelloWorldApp
```

References:

- <http://java.sun.com/docs/books/tutorial/getStarted/cupojava/win32.html#1>
- **Sun Microsystems, Introduction to Java program**

Appendix C:

Using SciTE

We are going to learn to use the simple text editor called SciTE (which stands for Scintilla Text Editor) in the lab, and you should find it quite easy to use it at home also. It is not strictly an IDE, but it has most of the features of one, in a simple package. Additionally, it can be used with compilers of programs written in several other languages, so it is useful to you in other ways later.

Instructions on downloading SciTE

- If you are using a Windows machine download Scite from:
<http://www.scintilla.org/SciTEDownload.html> click on the "Full Download" link under the "Windows" section.
- If you are using a Linux machine, you should click on the "Full Download" Link under the "GTK+ / Linux" section.

Once you are done with downloading JDK/JRE and SciTE, you can

- 1) use SciTE as a standalone editor for your java programs
- 2) integrate SciTE with a Compiler to have an editor and be able to Compile and Run Java programs from within SciTE.
For this purpose, copy the two files zzz.properties, langara.properties into the folder Documents\wscite443\wscite. This is where SciTE is installed by default. The SciTE application is in this directory as well and you can Pin it to your TaskBar.

Warning: SciTE has a most annoying “feature”: the backspace key is not usually implemented in the input console but it is buffered. Thus, if you are typing input into your program and you use the backspace key, the character just entered is not removed from the buffer.

Editing, Compiling, and Running programs using SciTE

SciTE is a simple text editor, though it has decent editing features, but it will enable you to compile and later execute Java programs from within it. It includes: a text editor, a compiler, and a debugger.

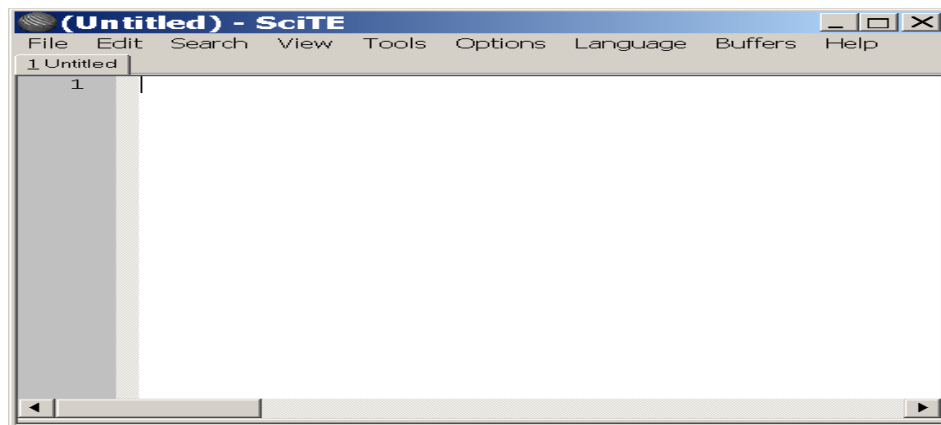
It is extremely important that you follow the instructions exactly.

- **Basic Concepts**

You will create a simple program that displays a message (Your name and a few other greeting messages) to the user. You will enter Java source code into a new file; save it in your folder for lab0 under the name **Greetings**, and then you will ask SciTE to compile your code. This will cause the following actions to occur.

1. When you do, the SciTE editor will invoke the Java compiler (a separate program) to read the file Greetings.java and check it for errors.
 2. If compilation is not successful, SciTE will show you, in a window in a lower part of the screen, the compiler's error messages that should help to indicate to you where problems are to be found.
 3. When compilation is successful, the compiler will then create a new file for you in the same directory called Greetings.class, this file contains a unique kind of machine language translation of your program called the Byte Code.
 4. Now you can ask SciTE to have Java execute your byte code. At that moment, your Greetings.class file will be converted into Windows executable code (perhaps after a small delay). When this happens, any output results (or further error messages) will be generated and placed into the same output window below your source program.
- **A Java Application –Greetings**
The **SciTE** application is accessible from any Lab computer.

When SciTE starts to run, you should see the following window appear:



Type the following into the SciTE editor window:

```
public class Greetings
{
    public static void main (String[] args)
    {
        System.out.println("Welcome to CPSC1150 course.");
        System.out.println("My name is YourFirstName YourLastName")
        System.out.println("My student ID is YourStudentID");
    }
}
```

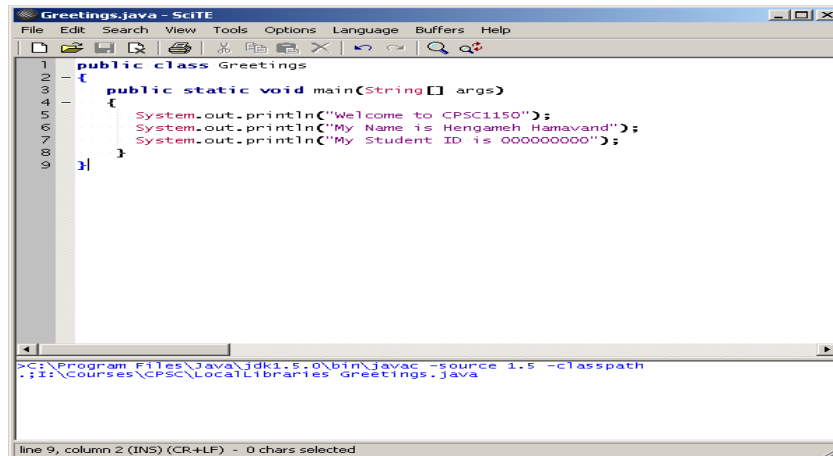
When you have done this, find and choose Compile on the Tools menu. You can notice that your file is saved each time you do this.

The first time there is no file name for it yet. You must name the file `Greetings.java` spelled exactly (including case) as it is in the source code above; and make sure it ends up in your `CPSC1150/lab0` subdirectory.

When you compile, you will have some error(s). Try again!

Bad news from the compiler is Exit code: 1, which means you have typos to fix.

Good news is Exit code: 0, which means no errors.



```

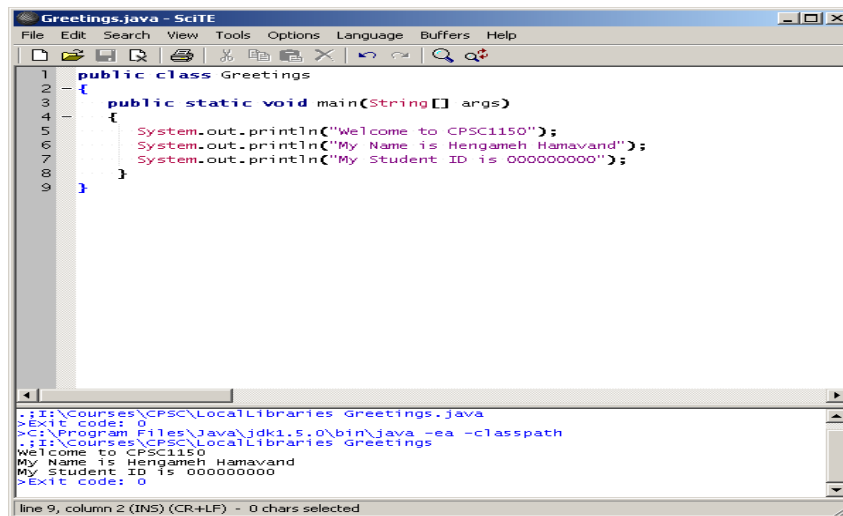
Greetings.java - ScITE
File Edit Search View Tools Options Language Buffers Help
1 public class Greetings
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Welcome to CPSC1150");
6         System.out.println("My Name is Hengameh Hamavand");
7         System.out.println("My Student ID is 000000000");
8     }
9 }

>C:\Program Files\Java\jdk1.5.0\bin\javac -source 1.5 -classpath
.:I:\courses\CPSC\LocalLibraries Greetings.java

line 9, column 2 (INS) (CR+LF) - 0 chars selected

```

As soon as your program successfully compiles, choose **Go** on the **Tools** menu. You should now be looking at the result as follows.



```

Greetings.java - ScITE
File Edit Search View Tools Options Language Buffers Help
1 public class Greetings
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Welcome to CPSC1150");
6         System.out.println("My Name is Hengameh Hamavand");
7         System.out.println("My Student ID is 000000000");
8     }
9 }

>I:\courses\CPSC\LocalLibraries Greetings.java
>Exit code: 0
>C:\Program Files\Java\jdk1.5.0\bin\java -ea -classpath
.:I:\courses\CPSC\LocalLibraries Greetings
Welcome to CPSC1150
My Name is Hengameh Hamavand
My Student ID is 000000000
>Exit code: 0

line 9, column 2 (INS) (CR+LF) - 0 chars selected

```