Langara College

Department of
Computing Science & Information Systems

Laboratory Guide[1]
CPSC 1150

[1] The document has been adopted and updated from other instructors in this course.

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 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. For this reason, you must prepare before you come to the lab. Typically the preparation includes an appropriate algorithmic solution for the current lab assignment. On the other hand, you should make use of the open lab hours whenever necessary.

1. General Instructions

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. Method of Submission

You must submit your completed assignment work to *BrightSpace (D2L) on the due dates*. A sample program **ready to be handed in 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.

3. 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.

3.1 External Documentation

Your external documentation should include:
1. the program's name
2. the purpose of the program

3. a list of other programs/packages, if any, that must be combined with this program, to make a workable combination
4. a list of classes, if any, involved in the program
5. the program's limitations (input it can't handle, a list of error messages that might be printed, round-off error)
6. a list of bugs you haven't fixed
7. the name of any program you borrowed ideas from
8. the **pseudocode** of how program works

3.2 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:

1. the program's name
2. the date you finished it
3. the purpose of the program (if necessary)
4. the meaning of each variable (if necessary)
5. any branching must be explained (if necessary)
6. the *purpose* of each class as well as specifications for each method including
   - *precondition* which is a statement on the conditions that must exist on the input parameters when the method is called
   - *postcondition* which is a statement of the conditions after the method is executed

Jave provides Javadoc to provide automatic documentation. To gain more information watch this and read this document.

4. 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.

5. 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.

5.1 Integrated Development Environments (IDEs)

Integrated program 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! At the top end come **JBuilder** and **CodeWarrior**, most often used by professionals with money.

Then there are free IDEs like **NetBeans** and **Eclipse**. These involve some effort to learn to use, but many people like them after they have survived the learning curve.
There are stripped down Java IDEs such as **SciTE.** Refer to **Appendix C** for an introduction to **SciTE**.

Finally, you can use a simple free text ediror like Sublime 3 to write your program. Then compile and run your code using the command prompt. This is the simplest and easyiest way for biginners, especially for those of you who have no experience of working with IDE before. This way reduces the complications and you only deal with language (Java) rather than dealing with langauage and IDE both at the same time. You can download Sublime to write your source code. Also you can find the commands to compile and run your code using command prompt in Appendix B.

6. 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. 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 program code and comments should be arranged so that the program is easy to read.
4. Use blank lines, space characters and tab characters in a program to enhance program readability.
5. 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.
6. 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.
7. Make sure that the braces { } are aligned.
8. 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.
9. Set convention for the indent size you prefer and then uniformly apply that convention. The *Tab* key may be used to create indents, but tab stops may vary between editors. We recommend using 3 to 5 spaces to form a level of indent.
10. 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.
11. Place a space after each comma in an argument list (,) to make programs more readable.

12. 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).
13. 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.
14. 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.
15. Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.
16. Using parentheses for complex arithmetic expressions even when the parentheses are not necessary, can make the arithmetic expression easier to read.
17. Indent the statement in the body of an ***if*** structure to make the body of the structure stand out and to enhance program readability.
18. Place only one statement per line in a program. This enhances program readability.
19. 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.
20. 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.
21. 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.**
22. Any assumption made by the program should be stated. For example, "the program assumes that the input data is valid".

By the way

Did you know that due to computer software flaws:

1. In January 1998, New York State computers erroneously declared the end of Medicare, the medical plan for senior citizens.
2. In March 1998, the new automated system of the Immigration and Naturalization Services (INS) erroneously declared thousands of people who were eligible as US residents to be illegal immigrants.
3. In April 1998, banks ATM machines stopped to release cash for cardholders for many hours in US.
4. In June 1998, the computer system in the Air Traffic Control (ATC) office of Long Island caused delays in the departure of hundreds of airplanes.
5. In September 1998, the engine of the Ukranian Zenith II Rocket was turned off immediately after take off causing 12 Telecom satellites, worth US $12,000,000 be destroyed.
6. In December 1998, the aircrafts of Northwest and Toronto collided together in Southwest Albany in New York.
7. In April 1999, the Internal Revenue Service computers of the USA stopped for 16 hours during the last hours of the deadline period for the tax returns.
8. In November 1999, Toshiba Company had to sign an undesired US $2.1 billion transaction.

Appendix A: (A sample program)

File AccountTest.java:

```
/**
 ** Program Name: AccountTest
 **@author          Student's Name
 **@since           JDK 1.8
 ** Course:         CPSC 1150
 ** Date:           May 7th, 2017
  */

public class AccountTest
{
        /**
         **  main: The main program which 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

                System.out.println("My name is " + name + ".");  //display the student's 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 0 if every thing is fine, otherwise it returns 1.
         **
         ** precondition: inputName is declared in the calling function.
         ** postcondition: all the information will be displays on the screen.
         */

        public static int 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 0; //return to where this function is called (note for void function return is optional).
        }
}
```

The above program has been documented suitable for Javadoc. It is a document generator tool in Java programming language for generating standard documentation in HTML format. It generates API

documentation. It parses the declarations ad documentation in a set of source file describing classes, methods, constructors, and fields. Learn more.

Sample External Documentation

Program AccountTest

File Name:	H:\CPSC1150\Lab0\ AccountTest.java
Purpose	To test the Account class and its methods.
Input	No input is required for this program.
Output	Student information displays on the computer screen

Program Logic **(Pseudocode)**

Algorithm AccountTest
START
1. Step1
2. .
3. .
4. .

END AccountTest.

Appendix B: Introduction to JDK

Java Development Kit (JDK) is a software development kit (SDK) for producing Java programs. The JDK is developed by Sun Microsystems.

Creating Your First Application

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

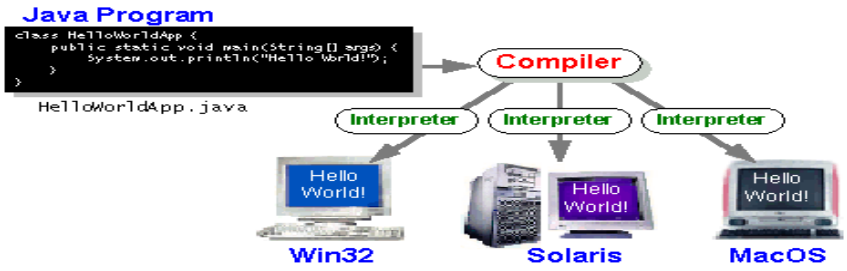
- **Create a source file.** A source file contains text, written in the Java programming language, that you and other programmers can understand.
- **Compile the source file into a bytecode file.** The *compiler*, javac, takes your source file and translates its text into instructions that the *Java Virtual Machine* (Java VM) can understand. The compiler converts these instructions into a bytecode file.

  javac HelloWorld.java
- **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.

  Java HelloWorld

Why Bytecodes

So, you've heard that with the Java programming language, you can "write once, run anywhere." This means that when you compile your program, you don't generate instructions for one specific platform. Instead, you generate Java bytecodes, which are instructions for the Java Virtual Machine (Java VM). If your platform--whether it's Windows, UNIX, MacOS, or an Internet browser--has the Java VM, it understands those bytecodes.



a. Create a Source File.

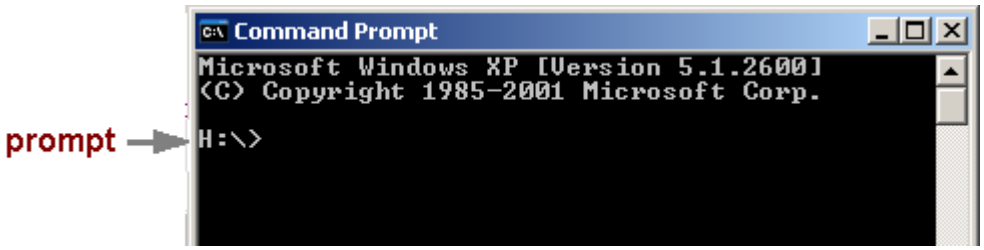
Start Sublime or Notepad. In a new document, type in the following code:
(Please note that you can use any other editor)

```
/**
 * 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:

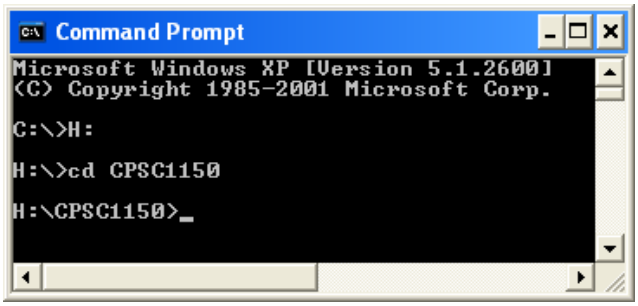


The prompt shows your *current directory*. To compile your source code file, change your current directory to the directory where your file is located. For example, if your source directory is CPSC1150\lab0 on the H drive, you would type the following command at the prompt and press Enter:

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

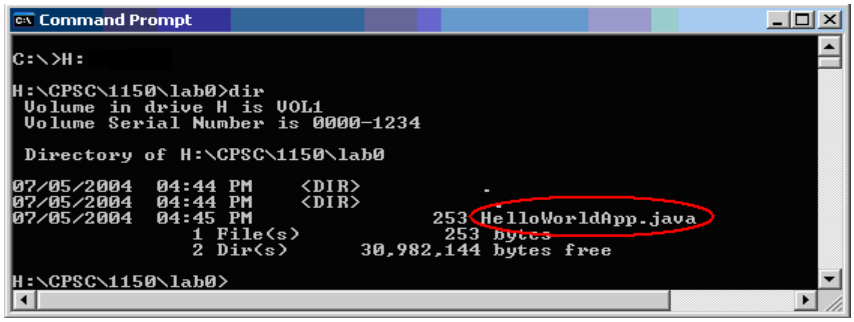
Now the prompt should change to H:\cpsc\1150\lab0>.

Note: To change to a directory on a different drive, you must type an extra command.



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 programmi**

Appendix C:

Using SciTE

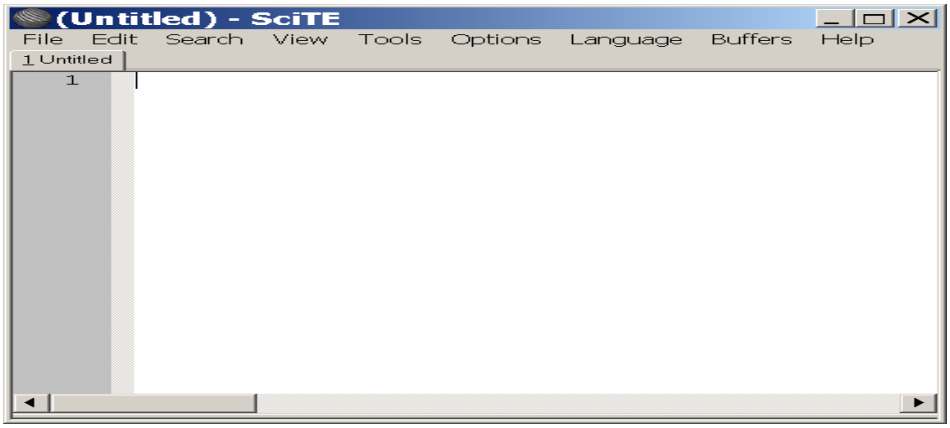
Editing, Compiling, and Running programs using SciTE

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.

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:

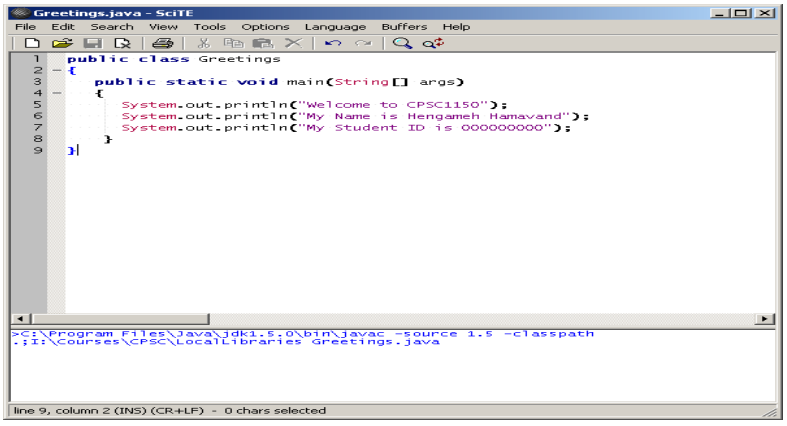


Type the following into the SciTE editor window:

```
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.prinlnt("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 H:\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.



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