

Session 1

Student Name _____

Other Identification _____

Introduction to Writing and Executing Java Programs

The purpose of this laboratory session is to introduce you to the computer system that you will use in the remaining sessions. The materials supplied during this period will teach you to:

1. Enter a Java program, compile it and execute it.
2. Find and correct compiling errors.
3. Investigate the `println()` and `print()` methods.

In preparation for this laboratory session, you should read Chapters 1 and 2 of *Computer Science: An Overview*.

Gaining Access to the Machine

Those of you using this manual may be working with a wide variety of computer equipment. While some may use individual machines, others may use machines that share resources over a network of remote terminals connected to a central computing facility that serves many users at once. No matter what the machine, your first contact with the computer will be through its operating system—a program that coordinates the activities of the machine and performs tasks as directed by the machine's user (or users).

You will most likely communicate with the operating system through a keyboard, mouse, and monitor screen. Through the machine's operating system, you will be able to activate numerous auxiliary programs, some of which are designed to assist in program development.

These are the programs you will learn to use in this laboratory session. In some cases these programs may be bundled as an integrated package; in other cases they may appear as individual programs whose services you must explicitly request. If you are using a school lab computer, your instructor or lab assistant can explain how your particular system operates. If you are using your own computer, you will need to download and install the software when directed.

High-Level Programming Languages

When we examine the internal design of a computer, we find that each individual operation the machine performs is tedious in nature and contributes only a minute step toward completing any meaningful task. These operations form a language known as the machine language. It is a language in the sense that these operations constitute a set of commands that the circuitry within the machine "understands."

To develop a program in the form of a machine language would be a laborious task. Thus, programs are normally developed using a high-level language that more closely resembles a human's natural language. Once the program has been written in this form, it is translated into a low-level machine language form so that the machine can execute it.

Today there are literally hundreds of high-level programming languages. You may have heard of Ada, FORTRAN, BASIC, C, C++, C# or COBOL. Some of these languages are designed for special types of programming tasks, while others are designed as general purpose programming languages. This manual will teach you the rudiments of the general-purpose language known as Java.

At Sun Microsystems in the mid 1990s, a team of programmers, headed by James Gosling, developed the Java programming language. Initial enthusiasm for Java was due to a new kind of program, called a Java *applet*. Applets are downloaded over the internet and run with the aid of a web browser. They can be animated and can interact with the user through a Graphical User Interface (GUI). Besides being a language for the internet, Java enjoys wide popularity in both the business and academic communities. In contrast to an applet, a Java *application* is a program that can run without the aid of a browser. Most of our programs will be Java applications, however, we will write applets in a later session.

Java is a very sophisticated and complex programming language. While this manual can only cover some of the features of Java, it does provide you with a working knowledge of the language, which you can expand in the direction of your own interests.

Object-Oriented Programming

Java is an object-oriented programming language. Object-oriented programming uses the concept of objects to mimic real life. Our world is filled with objects that interact with one another. For example, let's consider a class of objects called Car. If I tell you that "I just bought a new car," you would have a basic idea of what I purchased. A *Car* has both *characteristics* such as make, model, and color, and *behaviors* such as the ability to start, accelerate and stop. In object-oriented terminology we say, a **class** describes the characteristics and behaviors that objects of this type have. The class is not the same as the object. Rather, the class is a template from which objects can be created. Therefore, we say an **object** is an instance of a class. My car is an object that is a specific instance of the class Car. It is a new Car whose make is Mercedes, model is E740 and color is red.

Let's list some other examples of classes and objects that are created from those classes. I'll list three, and then you can list three.

Class	Object
CartoonCharacter	Mickey Mouse, Bugs Bunny, Tweedy Bird
Movie	Star Wars, Gone With the Wind, E.T.
Clock	my kitchen clock, my watch, your classroom clock

I can send messages to my car to invoke its behaviors. For example, inserting a key into the ignition and turning it clockwise sends a message to my car to start. When I want to send a message to my car to accelerate, I step on the gas pedal. Stepping on the brake pedal, sends the message to stop. What are some messages that you might send to an object that was created from the class Clock?

Object-oriented programming is a way of organizing a program. For example, consider a video game. Following the object-oriented style, all of the characteristics dealing with a figure on the screen, such as location, direction, and speed of motion, are described in a class. In addition, the class describes behaviors that allow the figure to interact with other components of the video game. From the class description, many figures can be created. The organization of the entire program would be a collection of objects that interact with one another.

Java provides us with a library of classes from which objects can be created. This library of classes is called the *Java Application Programming Interface*, or Java API. We will begin using some of these classes and objects in our first programs. You will learn to define your own classes and create your own objects in later laboratory sessions.

Don't worry if you don't totally understand all of this yet, you can still get through this session successfully. If you would like to learn more about object-oriented programming now, you can read ahead in Chapter 6 of *Computer Science: An Overview*. Otherwise, we'll be doing more with it in future sessions.

The Program Preparation Process

The steps required to develop programs using the Java language will depend on the computer installation being used. However, some features of the process are common to all systems. As a first step, the programmer uses a program called an editor to type a Java language version of the program being developed. This editor may be a stand-alone utility program or a part of an integrated software development package. Once the program has been typed, it is saved as a file with a `.java` extension in mass storage. This version of the program is known as the source program because it is the initial, or source, version of the program. It is this version to which you will return when corrections or alterations to the program are required.

A program in its source form cannot be executed by the computer. Traditionally, the program source code must first be translated into the machine's low-level language by a program known as a translator or compiler. However, the Java compiler translates the program source into a universal language called Java bytecode. To run a Java program, your computer must have a Java interpreter to interpret each bytecode statement into machine language. Most browsers contain a Java interpreter, (often called a Java Virtual Machine) that is used to execute applets. For Java applications, however, the computer must have a Java interpreter available elsewhere.

If you are using a school computer lab machine, your instructor or lab assistant can explain the details of how to type, save, compile, and finally execute programs using your school's particular computer system. If you are using your own computer, you will need to download the Java SE Development Kit (JDK) and an integrated development environment (IDE). JDK can be downloaded for free from <http://java.sun.com/javase/downloads/index.jsp>. You will want to make sure you download the JDK which includes the command line development tools, not just the runtime environment. After you download the file, you must install the JDK before installing an IDE. The next step is to download an IDE. There are several available at no cost. An easy to use IDE is jGrasp from Auburn University, available at www.jgrasp.org. A little more sophisticated choice is Eclipse, available at <http://www.eclipse.org/downloads/>.

Experiment 1.1

This experiment introduces the compiling process and the manner in which your compiler reports the errors that it finds. Error messages vary greatly from one system to another. Some systems are very helpful to the programmer, and some are not. Most Java compilers pinpoint the line where the error has occurred and also display a short error message. It is then up to you to play the part of detective, going back into the source program and searching for the error yourself. Often, one small error can spawn several misleading messages pointing to false errors. A common procedure when looking for errors (or "debugging") is to compile the program again after correcting the first real error that you find. It often happens that one correction will cause other misleading error messages to disappear.

Step 1. The following document is a simple program in the Java programming language. Using the editor in your particular software development environment, type the program as it appears here and save it, for future reference, in a file called `First.java`. Be careful to include all of the punctuation marks and the braces.

```
// This is our first Java program.
class First
{
    public static void main(String[] args)
    {
        System.out.println("What is your favorite");
        System.out.println("flavor of ice cream?");
    }
}
```

Step 2. Retrieve the program from Step 1 and compile it. If the compiler finds errors (which would be typing errors), correct them and try again. Execute the final, correct program. What happens?

Step 3. In case you didn't have any typing errors on your first attempt, we'll introduce some now. Change the word `System` in the source program to `system` and try to compile the altered version. How does your compiler inform you of this error?

Step 4. Correct the error introduced in Step 3, and then remove the semicolon from the end of the line

```
System.out.println("What is your favorite");
```

Try to compile this altered version. How does your compiler respond?

Step 5. Correct the error introduced in Step 4, and then remove the closing brace } at the end of the program. Try to compile this altered version. How does your compiler respond?

Step 6. Correct the error introduced in Step 5, and then change the spelling of `main` to `maine`. Compile and execute the program. What run-time error message did you get?

A Simple Java Program

Let's examine the program listed below, which is the program used in Experiment 1.1. Note the line numbering on the left hand side is a feature that can be turned on and off in your IDE.

```
#1      // This is our first Java program.
#2      class First
#3      {
#4          public static void main(String[] args)
#5          {
#6              System.out.println("What is your favorite");
#7              System.out.println("flavor of ice cream?");
#8          }
#9      }
```

In addition to statements that will ultimately be translated into machine language, a Java program can contain explanatory remarks for the aid of human readers. These remarks, known as comments, are placed after the characters `//`. In turn, the Java compiler ignores everything appearing on the same line after these special characters. In our example program, line #1 is a comment. Comments have a variety of uses. They can be inserted to clarify a section of a program that might otherwise be hard to understand. They can also be used to give information about the creation of a program, such as the date last modified and author.

All Java programs consist of one or more classes. Our example consists of only one class. In line #2, the class definition begins with the word `class` followed by the name `First`, which is the name by which the class will be identified. Line #3 is an opening brace `{` that defines the beginning of the body of the class. Line #9 is a related closing brace `}` that defines the end of the class body. We say the body of the class is *delimited* by a pair of braces. Notice that the two braces that delimit the body of the class are at the same indentation level. This is an attempt to make the program easier to read.

A *method* is a named group of statements that can be executed as a unit. Synonyms for method are function and procedure, but in object-oriented programming, *method* is the preferred term.

Every Java application program begins execution in a method named `main`. Line #4 contains the opening line, or *method header*, of the method `main`, which must always be:

```
public static void main(String[] args)
```

We'll learn more about method headers in a later laboratory session.

In lines #5 and #8, the body of the `main` method is delimited by a pair of braces that are at the same indentation level as the method header. The body of the method consists of two statements, in lines #6 and #7, which are executed when the program is run. Each statement is terminated by a semicolon. Although not mandatory, it is customary to place each statement on a separate line and to use indentation to help the reader identify these statements as part of the body of the `main` method.

The executable statements in lines #6 and #7 are similar. `System.out` is a predefined object that is used to send information to the monitor. To print to the monitor screen, we send the message `println` (read print-line or print-line) to the object `System.out`. In a Java program, a request to send a message to an object is expressed by the object name, followed by a period (which is called the dot operator), followed by the message being sent—hence: `System.out.println`. When a message is sent to an object, the method in the object whose name corresponds to the message is invoked. A parenthetical expression following a request to send a message to an object is used to enclose information that the method needs to execute properly. In our example, the parenthetical expression encloses the string of characters that should be displayed on the monitor screen. Note that the string of characters is delimited by a pair of double quotes.

When executed, the program will cause these two lines to appear on the screen.

```
What is your favorite
flavor of ice cream?
```

Experiment 1.2

Step1. Modify the original program in Experiment 1.1 to use the method `print()` instead of `println()`. Compile and execute the program. How does this output differ from the output of the original program?

```
System.out.print("What is your favorite");
System.out.print("flavor of ice cream?");
```

Step 2. Make a conclusion: What is the difference between `println` and `print`?

Step 3. Modify the existing program by adding `\n` at the end of each string as shown below. Compile and execute the program. How does the output differ from the output above? How does it differ from the output in the original Experiment 1.1?

```
System.out.print("What is your favorite\n");  
System.out.print("flavor of ice cream?\n");
```

Step 4. Add the following lines to the end of the main method. Compile and execute the program. How does the output produced by each of these statements differ from the others?

```
System.out.println("1:Chocolate,\nstrawberry, vanilla?");  
System.out.println("2:Chocolate,\nstrawberry,\nvanilla?");  
System.out.println("3:Chocolate,\n\nstrawberry,\n\nvanilla?");
```

Step 5. What does the `\n` character combination mean?

Experiment 1.3

Step 1. Replace the statements in the main method of Experiment 1.2 with the single statement below. Compile the new program. Record what happens.

```
System.out.println("Oh, I love to "program" in Java");
```

Step 2. Correct the statement in Step 1 so that it looks like the first statement below and add the second statement. Compile and execute the program. Record the results.

```
System.out.println("Oh, I love to \"program\" in Java.");  
System.out.println("Oh, I love \to \"program\" i\n Java.");
```

Step 3. Draw a conclusion: What is the meaning of the backslash mark? What are the meanings of \" and \t ?

Experiment 1.4

Step 1. Replace the statements in the main method in Experiment 1.3 with the statements below. Compile and execute. Record the results.

```
System.out.println("1:Send money quick!"); // To Mom!  
System.out.println("2:Send money quick!// To Mom!");  
// To Mom! System.out.println("3:Send money quick!");
```

Step 2. Add the following lines to the end of the main method. Compile and record the results. What rule can you derive about the placement of comments?

```
System.out.println("4:Send money quick!" // To Mom!);
```

Post-Laboratory Problems

1.1 . Write a program that prints the message

```
My name is Hector,  
    I am a vector.  
        I am the subject of many  
            a physics lecture!
```

- all on one line
- on two lines
- on four lines as printed above with indentation
- inside a box drawn with asterisks

1.2. Find the errors in the following program.

```
Class Second  
{  
    public static main()  
    {  
        System.out.println("I like to write before I've read it./n");  
        System.out.println(Then with my pen, I always edit.);  
        System.out.println("But, with computer\s, now I type);  
        System.out.println("And never, ever get it right"\n)  
    }  
}
```

1.3. What would be the output of the following program?

```
class Empty  
{  
    public static void main(String[] args)  
    {  
    }  
}
```