

Table of Contents

Introduction to Java Programming	2
Overview	2
Learning Objectives.....	2
About Java	2
Useful Information	3
Necessary Tools.....	3
Web References	3
Reference Books.....	3
Contact Information.....	3
Linux Essentials	4
A Sampling of Basic Linux Commands	4
Creating Your Program	5
Program Names	5
The Program Template	5
Program Structure	6
Statements.....	6
Code Blocks.....	6
Case Sensitivity and Whitespace	6
Your First Program – HelloWorld.java	7
Output Command: <code>System.out.println()</code>	7
The HelloWorld.java Program.....	7
Compiling and Executing	8
Compiling Your Program	8
Executing Your Program	8
Program Documentation	11
Comments	11
Calculations	13
Numbers	13
Operations.....	13
Memory	14
Storing Data in Memory	14
More on Memory	15
Variables.....	15
Identifiers.....	15
Declarations.....	15
Assignment Statements.....	15
Referencing Variables	16
Interactive Programs	19
Input Command: <code>UserInput.getType()</code>	19
Storing Input	19
Advanced Programming Constructs	23
Conditional Statement: <code>if (condition) { statements; }</code>	23
Conditional Statement: <code>else { statements; }</code>	24
Advanced Programming Constructs	27
Loops: <code>for (initialization; condition; increment) { statements; }</code>	27
Appendix A: UserInput.java Source Code	31
Appendix B: Java GUI (Graphical User Interface)	32

Introduction to Java Programming

Overview

Computer programming is *the process of planning and creating a sequence of steps for a computer to follow*. In general, this process will help us resolve a problem which is either too tedious or difficult to work out otherwise. In this class we will utilize the Java programming language, on a remote computer running the Linux operating system, to implement the actual steps.

Learning Objectives

Over the course of the class you will learn how to perform the following tasks:

- create Java *source code* with a text editor
- compile your source code – identify and fix any *programming* errors
- run and test the executable – identify and fix any *logical* errors
- insert program *documentation* where appropriate
- incorporate advanced programming constructs

About Java

Java is a full-featured programming language similar in functionality to C++ and other “high-level” languages. Once heralded as the next-best-thing for the web, Java is instead regarded as an exceptional language for creating stand-alone applications. This is in no small part due to the relative ease with which Java can create GUIs (Graphical User Interfaces).

Many Colleges and Universities now teach Java in “Computer Programming 101”. The High School AP (Advanced Placement) test in computer science will also be based on Java starting in 2003.

Useful Information

Necessary Tools

- Text Editor: pico or vi on Linux machines
- J2SE SDK: Java 2 Platform, Standard Edition Software Development Kit

Web References

The official Java™ web site from Sun Microsystems – downloads, documentation, and tutorials:

<http://java.sun.com/>

The official API Specification for the Java 2 Platform, Standard Edition, v 1.4.1:

<http://java.sun.com/j2se/1.4.1/docs/api/>

Reference Books

A comprehensive introductory text for beginning programmers:

Java How to Program, Fourth Edition by Deitel & Deitel – ISBN: 0-13-034151-7

A reference manual for experienced programmers:

Java in a Nutshell, Third Edition by Flanagan – ISBN: 1-56592-487-8

Contact Information

Eric Ovaska, LANL 505-667-1019 ovaska@lanl.gov

Linux Essentials

A Sampling of Basic Linux Commands

Linux is essentially Unix running on PC hardware. It often has a Graphical User Interface (GUI) similar to a Mac or PC. However, it is more common to interface with a Linux machine remotely via textual commands. On the Mode machine, note that the prompt will reflect which directory (folder) you are currently working in.

The following text-based commands will help you to get started:

ls – *lists* the names of the files and sub-directories inside the current directory

- **ls -F** – explicitly indicates which items are directories (folders) by appending a forward slash to the directory name
- **ls -l** – supplies *lots* of information regarding your files and directories

cd – changes your current working **d**irectory

- **cd ..** – moves you into the parent directory
- **cd *directory*** – moves you into the subdirectory named *directory*

pico – invokes a menu-driven text editor for creating/editing files

- **pico *filename*** – edits (or creates) a file named *filename*

javac – invokes the Java compiler

- **javac *program.java*** – compiles the Java source code file named *program.java*, and produces an “executable” file named *program.class*

java – invokes the JVM (Java Virtual Machine), which executes the program

- **java *program*** – executes the program instructions in the file *program.class*

At this time, connect to *mode.lanl.k12.nm.us* with your assigned login names and password. Experiment with the above commands.

Creating Your Program

Program Names

Every file containing a Java program must end in “.java”, with the rest of the name containing nothing more than letters, numbers, and the underscore character. By convention, all program names in Java begin with a capital letter – each “word” in the name is capitalized as well.

- No *spaces* in the filename!
- Make the name as short and descriptive as possible

The Program Template

Every program you create on Mode should contain, at a minimum, the following code:

filename: **Template.java**

```
import gov.lanl.java.UserInput;

public class Template
{
    public static void main (String args[])
    {
        Your Statements Here;
        Your Statements Here;
        Your Statements Here;
        Your Statements Here;
        System.exit(0);
    }
}
```

Note: you must always match the *prefix of the filename* and the *class name*!

The *statements* comprising your program instructions will be typed into this template between the inner pair of curly braces – this area is the main *code block*. Your instructions will also appear before the statement “System.exit(0);”

Typically, your statements are executed from top to bottom. We will see exceptions of this later on ...

Program Structure

Statements

A statement represents a single step in your program – it may be relatively complex at times

- Always place a semicolon after a statement
- In general, try to place statements on individual lines within your program

Code Blocks

A code block is a collection of statements contained within a pair of curly braces

- In general, do not place semicolons after “block headers” – the statement preceding a code block

Case Sensitivity and Whitespace

Be very mindful of how you type in your program – Java is a *Case Sensitive* language!.

Furthermore, Java is a *free-format* language. This means that most combinations of *whitespace* (spaces, tabs, carriage returns, etc.) are ignored.

Note: The above program (Template.java) could thus be written entirely on one line! It is, however, better to structure your code in such a way that it is visually easy for you and others to interpret the source code.

Your First Program – HelloWorld.java

Output Statement: `System.out.println()`

A computer program is useless if we cannot obtain information back from it. It is therefore essential that we be able to incorporate a command which will allow the program to tell us what it knows.

`System.out.println(information)` – this command will output your *information* to the screen, followed by a carriage return.

- The *information* may be a sequence of characters between double quotes
 - Will display literally as entered between the quotes
- The *information* may also be a *variable* – more on this later ...

The HelloWorld.java Program

Enter the following code using **pico** into a file named *HelloWorld.java*.

filename: **HelloWorld.java**

```
import gov.lanl.java.UserInput;

public class HelloWorld
{
    public static void main (String args[])
    {
        System.out.println("Hello World!");
        System.exit(0);
    }
}
```

Changes from *Template.java*:

- File/class name (notice how all of the “words” are capitalized)
- One unique statement – an output command

Now what do we do with it? How do we actually execute (run) the program? Use the tools in the J2SE SDK!

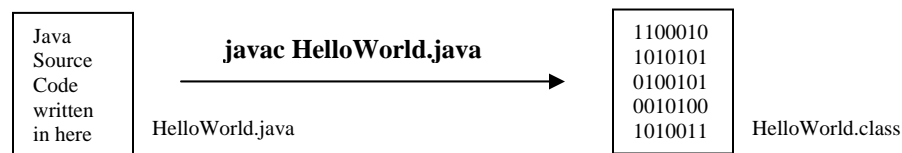
Compiling and Executing

Compiling Your Program

The instructions within your source file are only intelligible to you – not the computer. i.e. the computer does not understand English words and phrases!

You must use the **javac** command to invoke the *java compiler*:

- Any *programming errors* will be reported at this time
 - Typically syntactical or typing errors
- If there are no errors, translates the source code to “byte code”
 - A file named *program_name.class* will be created

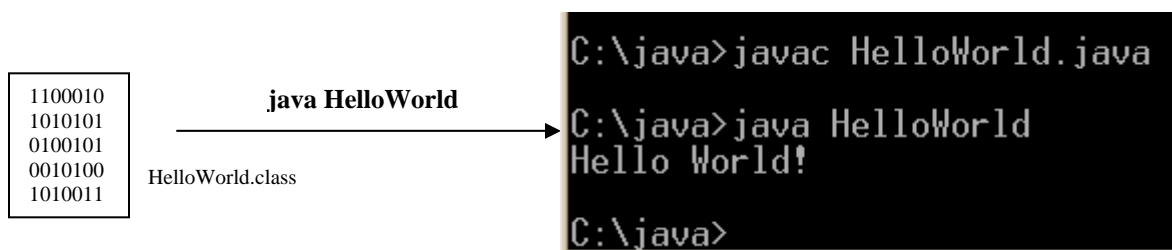


Executing Your Program

The instructions inside of your “.class” file are (almost) in a language the computer truly understands.

To execute your program, use the *java* command to invoke the **java interpreter** (JVM):

- Interprets the byte code at the “system level”, and executes your instructions



Now you can compile and execute *HelloWorld.java*! Watch for errors ...

Exercise 1

Create an *Address Label* Program for yourself which produces output similar to below.

```
C:\java>java AddressLabel
Eric Ovaska
3959 Trinity Drive
Los Alamos, NM
87544
C:\java>
```

Question:

- Try using `System.out.print()` instead of `System.out.println()` in some of your statements – what is the difference?

Exercise 1 – Solution

filename: **AddressLabel.java**

```
import gov.lanl.java.UserInput;

public class AddressLabel
{
    public static void main (String args[])
    {
        System.out.println("Eric Ovaska");
        System.out.println("3959 Trinity Drive");
        System.out.println("Los Alamos, NM");
        System.out.println("87544");
        System.exit(0);
    }
}
```

Program Documentation

Comments

You may include notes to yourself (or others who may be working on the program) from within your source code in the form of *comments*.

The compiler completely ignores any comments during the source-code translation. The following code demonstrates the use of both *single-line* comments and *multiple-line* comments:

filename: **HelloWorld.java**

```
import gov.lanl.java.UserInput;
/* This program created by Eric Ovaska
   on September 25, 2001 */

public class HelloWorld
{
    public static void main (String args[])
    {
        System.out.println("Hello World!");
        System.exit(0); // This line stops the JVM
    }
}
```

- Single-line comments
 - Compiler ignores all information between two forward slashes (//) and the end of the line – the slashes are ignored as well
- Multiple-line comments
 - Compiler ignores all information between a forward slash and asterisk (/*) and an asterisk and forward slash (*//)

Exercise 2

Record the file size (in Bytes) of the following files in your *java* folder:

- HelloWorld.java
- HelloWorld.class

Liberalily place comments throughout your source code. Once again compile and execute this program.

```
C:\java>java AddressLabel  
Eric Ovaska  
3959 Trinity Drive  
Los Alamos, NM  
87544  
C:\java>
```

Question:

How did the file sizes of “HelloWorld.java” and “HelloWorld.class” change?

Calculations

Numbers

Java can manipulate and store many *types* of data (information) in the memory of the computer. We will only be concerned with two numeric types: **int** (*whole numbers*) and **double** (*decimal numbers*).

- Any number typed into an equation with a decimal point will be considered to be of type *double* – without a decimal point of type *int*

Operations

Java can perform arithmetic operations, using the four standard operators of addition (+), subtraction (-), multiplication (*) and division (/). Java also follows the standard algebraic *order of operations*.

Be aware of performing calculations with different data types. Note the results of performing arithmetic operations with the following mixed data types:

- int & int = int
 - $5 + 4 = 9$
 - $4 / 5 = 0$
- int & double = double
 - $4 / 5.0 = .80$
 - $3.2 * 7 = 22.4$
- double & double = double
 - $5.3 / 8.5 = .6235294117647059$

Once a computer performs a calculation, where does it store the results?

Where does a human store information?

Memory

Storing Data in Memory

A program can store, among other things, the results of an arithmetic expression into the memory of the computer. This process typically involves using two statements:

1. **Declaration Statement:** Reserves a small portion of memory, called a *variable*, to use in storing data
 - Indicate what type of *data* will be stored there
 - Give the memory location a descriptive *reference name*
2. **Assignment Statement:** Places your *data* into the *variable*

You may access the information in the variable simply by referring to the specific *reference name*.

filename: **Addition.java**

```
import gov.lanl.java.UserInput;

public class Addition
{
    public static void main (String args[])
    {
        double sum; // declaration statement

        sum = 7 + 5; // assignment statement
        System.out.print("The Result Is: ");
        System.out.println(sum); // printing a variable
        System.exit(0);
    }
}
```

More on Memory

Variables

A *variable* is a dedicated piece of memory in a computer, where a program may store data during program execution.

Identifiers

An identifier is a *reference name* which is used for referring to a particular variable. You may use any “word” which contain numbers, letters, and the underscore character to form your identifier. By convention, all identifiers in Java begin with a lowercase letter – thereafter, each additional “word” in the identifier is capitalized.

Declarations

A declaration statement has the following form:

data_type identifier;

From this statement, we have reserved a location in memory (a *variable*) which may contain data of the type *data_type*, and may be referenced later by using the *identifier*.

Assignment Statements

To place data into a variable, you must use an assignment statement. An assignment statements has two parts; the left hand side with the *variable* name, and the right hand side with an *expression* to be evaluated. The two parts are separated by an equals (=) sign.

The expression is evaluated and the result is placed into the variable referenced by the identifier.

The evaluated expression *should* be of the same data type as the identifier on the left hand side references!

Referencing Variables

To access the data in a variable, you may simply use the *identifier* that references that particular memory location.

Note: In order to output the data in your variable, it will be necessary to use the *identifier* within a `System.out.println(information)` statement. The identifier will be the *information*

Memory does not need to be used just for storing the results of calculations – it can very readily be used to make your program more readable and flexible!

filename: Addition2.java

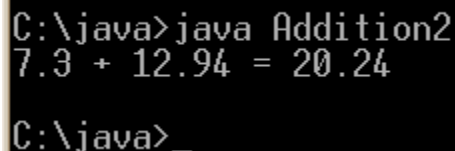
```
import gov.lanl.java.UserInput;

public class Addition2
{
    public static void main (String args[])
    {
        double firstNum, secondNum; // notice multiple declarations
        double sum;

        firstNum = 7.3; // this is our input
        secondNum = 12.94; // so is this ...

        sum = firstNum + secondNum; // this will be our output

        System.out.print(firstNum);
        System.out.print(" + ");
        System.out.print(secondNum);
        System.out.print(" = ");
        System.out.println(sum);
        System.exit(0);
    }
}
```



```
C:\java>java Addition2
7.3 + 12.94 = 20.24
C:\java>
```


Exercise 3

Create a program which will convert a given temperature in degrees Celsius to the corresponding temperature in degrees Fahrenheit.

```
C:\java>java Temperature
12 degrees Celsius = 53.6 Fahrenheit!
C:\java>
```

Use variables to represent both your input (degrees Celsius) and your output (degrees Fahrenheit). Furthermore, use an “int” variable for the input and a “double” variable for the output.

The formula to convert degrees Celsius to degrees Fahrenheit is as follows:

$$^{\circ}\text{F} = 9 / 5 * ^{\circ}\text{C} + 32$$

Hints:

- Be mindful when using the formula to calculate Fahrenheit; make sure you watch your use of “int” and “double” numeric data types!

Question:

- Are you more likely to find a temperature of 41° Celsius in Phoenix, Arizona or Anchorage, Alaska?

Exercise 3 – Solution

filename: **Temperature.java**

```
import gov.lanl.java.UserInput;

public class Temperature
{
    public static void main (String args[])
    {
        int celsius;
        double fahrenheit;

        celsius = 12;
        fahrenheit = 9.0 / 5 * celsius + 32;

        System.out.print(celsius);
        System.out.print(" degrees Celsius = ");
        System.out.print(fahrenheit);
        System.out.println(" Fahrenheit!");
        System.exit(0);
    }
}
```

Interactive Programs

Input Command: `UserInput.getType()`

Suppose we want to use our “Addition2” program multiple times. Each time we would have to edit the source code, re-compile, re-execute the program, edit the source code, etc.

There is an easier way to make your program reusable to those who do not have the knowledge to alter the source code!

- `UserInput.getType(message)` – this command will prompt the user with a *message* requesting input. The user input should be of the specified data *Type*;
- The *message* must be a sequence of characters between double quotes – will display literally as entered between the quotes
 - `UserInput.getInt(message)` – for requesting a whole number
 - `UserInput.getDouble(message)` – for requesting a decimal number

*** Note that these commands are only available for you to use if you have a copy of the `UserInput.class` file in on your system!*

Storing Input

The computer memory may also be used to store the data a user supplies to us as input to the program. Once again, we use an assignment statement to achieve this task:

```
firstNum = UserInput.getDouble("What is the first number? ");
```

Remember: *In an assignment statement, the expression on the right is evaluated first, and the result is placed in the variable referenced by the identifier on the left.*

This would effectively grab the input from the user, and place it into the memory location named “firstNum”.

The Addition3.java Program – Interactive!

filename: **Addition3.java**

```
import gov.lanl.java.UserInput;

public class Addition3
{
    public static void main (String args[])
    {
        double firstNum, secondNum, sum;

        firstNum = UserInput.getDouble("What is the first number? ");
        secondNum = UserInput.getDouble("What is the second number? ");

        sum = firstNum + secondNum; // this will be our output

        System.out.print(firstNum);
        System.out.print(" + ");
        System.out.print(secondNum);
        System.out.print(" = ");
        System.out.println(sum);
        System.exit(0);
    }
}
```

```
C:\java>javac Addition3.java

C:\java>java Addition3
What is the first number? 5.9
What is the second number? 3.3214
5.9 + 3.3214 = 9.2214000000000001

C:\java>
```

Exercise 4

Turn your temperature conversion program into an interactive program in which the user is prompted for the temperature in degrees Celsius he/she wishes to have converted into degrees Fahrenheit!

```
C:\java>java Temperature2
What is the temp in Celsius? 12
12 degrees Celsius = 53.6 Fahrenheit!
C:\java>
```

Hints:

- Be mindful when using the formula to calculate Fahrenheit; make sure you watch your use of “int” and “double” numeric data types!

Questions:

- How long does it take you to fill in the following table?

Conversion Table

°C	°F
-5	
0	
25	
50	
75	
100	

- What happens when you try to use a decimal value as input?

Exercise 4 – Solution

filename: **Temperature2.java**

```
import gov.lanl.java.UserInput;

public class Temperature2
{
    public static void main (String args[])
    {
        int celsius;
        double fahrenheit;

        celsius = UserInput.getInt("What is the temp in Celsius? ");
        fahrenheit = 9.0 / 5 * celsius + 32;

        System.out.print(celsius);
        System.out.print(" degrees Celsius = ");
        System.out.print(fahrenheit);
        System.out.println(" Fahrenheit!");
        System.exit(0);
    }
}
```

Advanced Programming Constructs

Conditional Statement: `if (condition) { statements; }`

By default, the commands in our main code block are executed top to bottom. With a conditional statement, the code block associated with the `if` clause will only execute if the `condition` evaluates to “true”.

- For our purposes, we will look at *conditions* that involve numerical comparisons

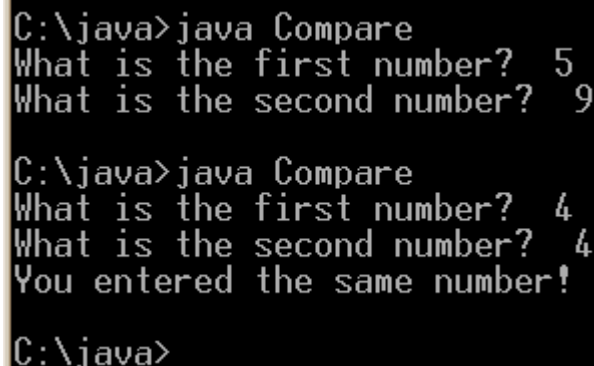
You may use the following numerical comparison operators to form a *condition* that will evaluate to either *true* or *false*

- `>` “greater than”
- `>=` “greater than or equal to”
- `==` “equal to”
- `!=` “*not* equal to”
- `<=` “less than or equal to”
- `<` “less than”

The following snippet of code from a program named “Compare.java” demonstrates the use of the `if` statement:

```
firstNum = UserInput.getInt("What is the first number? ");
secondNum = UserInput.getInt("What is the second number? ");

if (firstNum == secondNum)
{
    System.out.println("You entered the same number!");
}
System.exit(0);
```



```
C:\java>java Compare
What is the first number? 5
What is the second number? 9

C:\java>java Compare
What is the first number? 4
What is the second number? 4
You entered the same number!

C:\java>
```

Conditional Statement: `else { statements; }`

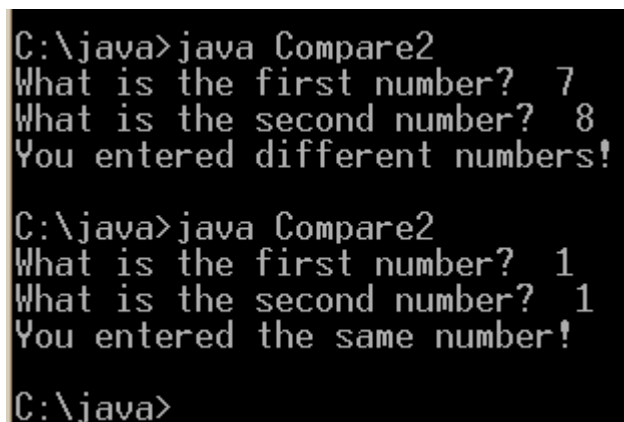
With a conditional statement, the code block associated with the *if* clause will only execute if the *condition* evaluates to “true”. A code block associated with an *else* clause will only execute if the condition evaluates to “false”.

- An *if* statement may or may not be followed by an *else* statement depending upon whether or not the program requires it
- An *else* statement never appears alone – it always follows immediately after the code block associated with an *if* statement
- There is no need to repeat the *condition* next to the *else* statement

The following snippet of code from a program named “Compare2.java” demonstrates the use of the *if* statement:

```
firstNum = UserInput.getInt("What is the first number? ");
secondNum = UserInput.getInt("What is the second number? ");

if (firstNum == secondNum)
{
    System.out.println("You entered the same number!");
}
else
{
    System.out.println("You entered different numbers!");
}
System.exit(0);
```



```
C:\java>java Compare2
What is the first number? 7
What is the second number? 8
You entered different numbers!

C:\java>java Compare2
What is the first number? 1
What is the second number? 1
You entered the same number!

C:\java>
```


Exercise 5

You own a theater which will be showing a “PG-13” movie. Create a “Movie Ticket” program which queries the user for their age, and allows them to enter only if they are at least 13 years old.

```
C:\java>java MovieTicket
How old are you? Don't lie ... 16
Come on in! Enjoy the show.

C:\java>java MovieTicket
How old are you? Don't lie ... 12
Go away you punk kid!

C:\java>
```

Hints:

Remember that you do not need semicolons after “block headers”.

Questions:

- What happens if the person is 13 years old?
- What if we only want to allow individuals who are teenagers (i.e. age 13-17)?

Exercise 5 – Solution

filename: **MovieTicket.java**

```
import gov.lanl.java.UserInput;

public class MovieTicket
{
    public static void main (String args[])
    {
        int age;

        age = UserInput.getInt("How old are you? Don't lie ... ");

        if (age >= 13)
        {
            System.out.println("Come on in! Enjoy the show. ");
        }
        else
        {
            System.out.println("Go away you punk kid!");
        }

        System.exit(0);
    }
}
```

Advanced Programming Constructs

**Loops: `for (initialization; condition; increment)`
`{ statements; }`**

By default, the commands in our main code block are executed top to bottom. With loop statements, the associated code block will execute over and over as long as the *condition* evaluates to *true*.

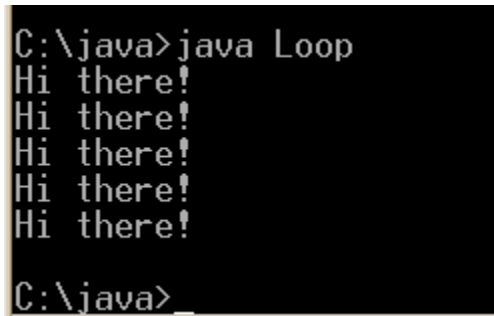
- The *for* statement is typically used when you want to execute a certain block of code a specified number of times.

When a *for* construct is encountered during program execution, the following steps are executed:

1. The *initialization* statement is executed – this might set an *int* variable to a value of 1
2. The *condition* (often involving the *int* variable) is evaluated. If it is “true”, then the *statements* within the associated code block are executed. If it is “false”, program execution continues immediately below the associated code block.
3. After the last statement is executed, the *increment* statement is executed – this might increase the value of the *int* variable by 1.
4. Steps 2 and 3 are repeated.

The following snippet of code from a program named “Loop.java” demonstrates the use of the if statement:

```
for (int i = 1; i <= 5; i = i + 1)
{
    System.out.println("Hi there!");
}
System.exit(0);
```



Notice how the initialization statement both declares an *int* variable named “i” and assigns a value of 1 to the memory location – all in one step!

Exercise 6

The following snippet of code will produce a randomly generated *int* value between 1 and 100:

```
1 + (int) (Math.random() * 100)
```

Create a “lottery” program which prints out a lottery ticket containing 5 randomly generated numbers with values between 1 and 100 – use a “for” loop to considerably shorten your program!

```
C:\java>java Lottery
Your lottery ticket numbers:
47 92 36 8 28
C:\java>
```

Hints:

You may give a variable a new value as many times as you like – each assignment statement removes the old value (if there was one) and places in the new value!

Questions:

- You win the lottery if you have the numbers 4, 14, 73, 91, and 1 – did you win?

Exercise 6 – Solution

filename: **Lottery.java**

```
import gov.lanl.java.UserInput;

public class Lottery
{
    public static void main (String args[])
    {
        int randomNumber;

        System.out.println("Your lottery ticket numbers:");

        for (int i = 1; i <=5 ; i=i+1)
        {
            randomNumber = 1 + (int) (Math.random() * 100);
            System.out.print(randomNumber);
            System.out.print(" ");
        }

        System.out.println();
        System.exit(0);
    }
}
```

Appendix A: userInput.java Source Code

/* userInput.getInt() and userInput.getDouble() will only work if you have a compiled version of this program in the same folder as the one that requires these functions */

```
import java.io.*;

public class userInput {

    public static int getInt (String message) {

        int intInput = 0;
        String userInput = null;
        // prompt the user to enter data
        System.out.print(message + " ");

        // open up standard input
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        // read the data from the command-line; need to use try/catch with the
        // readLine() method
        try {
            userInput = br.readLine();
        } catch (IOException ioe) {
            System.out.println("IO error trying to read input!");
            System.exit(1);
        }
        // convert user input to an integer
        try {
            intInput = Integer.parseInt(userInput);
        } catch (NumberFormatException nfe) {
            System.out.println("Error: Input provided is not a valid Integer!");
            System.exit(1);
        }
        return intInput;
    }

    public static double getDouble (String message) {

        double doubleInput = 0;
        String userInput = null;
        // prompt the user to enter data
        System.out.print(message + " ");

        // open up standard input
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        // read the data from the command-line; need to use try/catch with the
        // readLine() method
        try {
            userInput = br.readLine();
        } catch (IOException ioe) {
            System.out.println("IO error trying to read input!");
            System.exit(1);
        }
        // convert user input to a double
        try {
            doubleInput = Double.parseDouble(userInput);
        } catch (NumberFormatException nfe) {
            System.out.println("Error: Input provided is not a valid Double!");
            System.exit(1);
        }
        return doubleInput;
    }

} // end of userInput class
```

Appendix B: Java GUI (Graphical User Interface)

Graphical Output: `JOptionPane.showMessageDialog()`

Previously we had used a `System.out.println()` command to obtain text-based output. A relatively simple command is also available for graphical output

`JOptionPane.showMessageDialog(null, information)` – this command will display a small graphical dialog box with the *information* displayed

- The *information* may be a combination of quoted messages (displayed literally as typed in between the quotes) and variables
- When mixing quoted messages and variables, the two items must be joined together with a plus symbol (+), known in this context as the *concatenation operator*

filename: **AdditionGraphic.java**

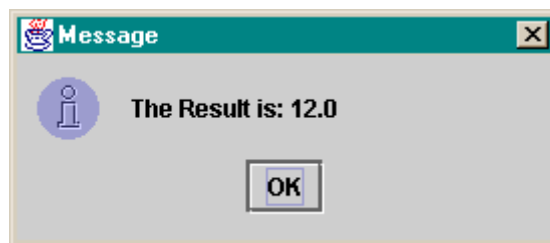
```
// **** Note the new import statement! ****
import javax.swing.JOptionPane;

public class AdditionGraphic
{
    public static void main (String args[])
    {
        double sum;
        sum = 7 + 5;

        // The old way ...
        // System.out.print("The Result Is: ");
        // System.out.println(sum);

        // The new way ...
        JOptionPane.showMessageDialog(null, "The Result is: " + sum);

        System.exit(0);
    }
}
```



Note that using successive `JOptionPane.showMessageDialog()` commands will not display the *information* from each command in the same dialog box – a new box will appear for each individual command!

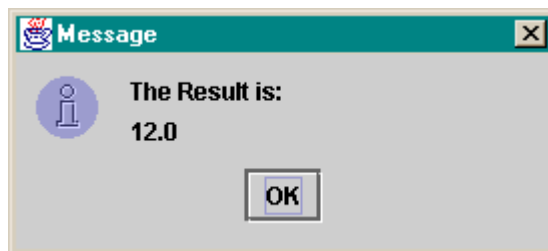
Using the concatenation operator (+) is thus essential for displaying a large message within one dialog box.

Furthermore, the information must contain at least one quoted message ... **variables alone will not work.**

Also note that the sequence of characters “\n” (without the quotes) will produce a newline character whenever it is encountered within a quoted message. In the previous program example,

```
JOptionPane.showMessageDialog(null, "The Result is:\n" + sum);
```

would produce:



Graphical Input: `JOptionPane.showInputDialog()`

Previously, we had used `UserInput.getInt()` or `UserInput.getDouble()` to obtain input from the user. A slightly more complicated command is available for graphical input.

`JOptionPane.showInputDialog(message)` – this command will display a small graphical dialog box with a *message* requesting input from the user.

- The *message* must be a sequence of characters between double quotes – will display literally as entered between the quotes
- The input is not considered an *int*, or *double*, but rather a *String* – a new data type which is nothing more than a sequence of characters
 - must be converted to *int* or *double* for later use
 - note the “S” in *String* must be capitalized

Converting Strings to Numbers

There are two useful commands to convert our *String* input to an *int* or *double* data type:

- **`Integer.parseInt(String)`**
 - This command will convert the *String* into an *int* (whole number)
- **`Double.parseDouble(String)`**
 - This command will convert the *String* into a *double* (decimal number)

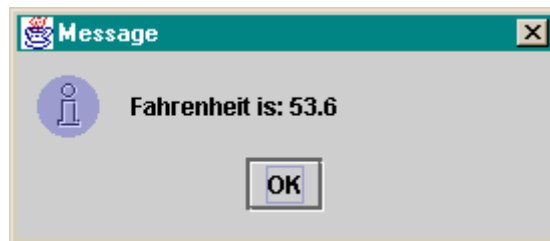
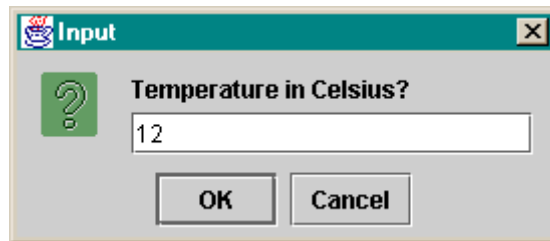
After we have converted our *String* into a numerical data type, we may then perform arithmetic calculations as we had previously.

filename: **TemperatureGraphic.java**

```
import javax.swing.JOptionPane;

public class TemperatureGraphic
{
    public static void main (String args[])
    {
        String celsius1; // Note the String variable
        int celsius2;
        double fahrenheit;

        celsius1 = JOptionPane.showInputDialog("Temperature in Celsius?");
        celsius2 = Integer.parseInt(celsius1);
        fahrenheit = 9.0 / 5 * celsius2 + 32;
        JOptionPane.showMessageDialog(null, "Fahrenheit is: " + fahrenheit);
        System.exit(0);
    }
}
```



Keep in mind that it is difficult to display large amounts of output in this manner ... more advanced GUI programming techniques must be learned in order to handle more complex situations.