Java Programming Applications

Author: Michael Robinson

The purpose of this writing is to show you how to learn Java Programming by implementing computer programming concepts using the Java Language, no previous programming knowledge is required.

Java has over 3.6 billion installations, as of this writing.

Humans communicate with each other by using thousands of languages. We communicate by sound, writing, signing and visually. Human languages have sets of rules that tell us how to use them so that we can understand each other.

All Computers, also known as Digital Machines, use a common numerical system to communicate with humans and each other, called The Binary Numerical System. Today's computer languages are implemented using plain English which in turn gets converted into Binary by other programs called Compilers and Interpreters.

We are living in a wonderful time where we can use multiple computer languages to solve our tasks, without having to worry about writing our programs in binary, which is quite complex and time consuming.

There are hundreds of computer languages, each having specific purposes with very clear and precise rules, and just like in human languages when we learn to communicate an idea with each other, if we need to do it in a different language all we need to learn is the syntax, we already know what we want to communicate (the concepts).

Once we learn one programming language like Java, writing our ideas in another language is easy because the syntax we use in many different languages is very similar, and in some cases exactly the same e.g:

```
if( temperature > 150 )
    exit
```

The world's first computer programmer was Lady Ada Lovelace, an English writer (10th December 1815 – 27th November 1852). Her work on Charles Babbage's analytical engine, an early mechanical general-purpose computer, is recognized as the first algorithm to be processed by a machine.

The computer language Ada, created for the United States Department of Defense, was named after Ada Lovelace.

This writing is intended to help my students understand the material that I teach in the classroom.

Learning Computer Programming to me is the same as learning to drive an automobile, it must be done by doing it (writing programs). In this writing the sequence of topics are presented in a logical manner, all concepts and commands in any section are either new or has been learned in a previous section.

All sample programs presented are original, and are to be used as an example for the programs that will be assigned as projects and/or exams.

To me writing a program is like building a house, and that is precisely how this writing has been created. First we start with the description of the task at hand, then we write the plans (pseudo code), and if needed a flowchart, following by the actual programing code called “source code”, finally we test it and then we make the guaranteed corrections. Pseudo code and flowcharts are done in class.

Rule number one in Computer Science says: “If a program works the first time, there is something wrong with it”.

Just like building a house, in programming, if we have a clear and precise description, pseudo-code, flowcharts, and knowledge of the language, creating the program is easier.

Usually people who start programing without the proper plans, spend great amounts of time fixing the errors (debugging the program).

Planning your work well, makes for an easy implementation of your plan.

Happy Programming!!!

-Michael Robinson

January 3rd, 2012

Content

Intermediate Java

Chapter One : The Basics

What is Java

Java has over 3.6 billion installations, as of this writing.

The programming language Java is known as “write once run everywhere” This means that we can write our programs on any device, convert it into a standard code and run it in any device that contains a JVM (Java Virtual Machine). Installing JVM in devices and software packages such as Windows has become an industry standard practice.

In theory we can expect to write a program in a Windows PC and run it on a Java enable microwave, cell phone, car, refrigerator, routers, mainframes, etc. without making any modifications. I have found that programs developed in Linux work in Windows and vice versa. However since there are multiple JVM implementations, that run on many different systems such as Windows, Linux, Mac-OS, HP-UX, Netware, Solaris, etc. there can be small differences in how a program executes, requiring changes to the program. There is a popular saying about Java among Java developers “Write Once, Debug Everywhere”.

What do we need

- A computer, any computer running Windows or Linux.

- The Java JDK
  The **Java Development Kit** (**JDK**) is an Oracle Corporation product aimed at Java developers.
  The JDK has as its primary components a collection of programming tools for Java.
  Download Java JDK from (FREE):
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
  Make sure you download the proper version for your computer.
  Windows 32 or 64 bit.

- An editor
  Is a program that allows you to create your Java programs.
  You can use an IDE or a text based editor.
  IDE Editors:
  - NetBeans
    Download NetBeans from (FREE):
    http://www.oracle.com/technetwork/java/javase/downloads/index.html

  - Eclipse
    Download Eclipse from (FREE) :

http://www.eclipse.org/downloads/moreinfo/java.php

Text Editors
- EditPro, Notepad, gedit

DO NOT use a word processor to create your programs.

What is an Operating System

Windows, Linux, and the Apple Operating system are the most popular ones. An Operating System is simply a set of programs that control your computer, they act as traffic cops.

What is a Computer Program Language.

Just like English, Spanish, French, etc are human languages used by humans to communicate with each other, Computer Programming Languages such as Java, C, Perl, PHP, etc are used by humans to give instructions to computers. Computers do what we tell them to do through Programming Languages.

There are two general types of Computer Languages, compiled and interpreted languages. A compiled language is a language that when compiled gets translated into a type of code called binary that the computer can understand. Binary is the natural language of computers, Compiled Languages are those like C, Cobol, Assembly, etc.

An Interpreted Languages are those that get translated into binary code when the program runs. Java is an interpreted language.

Since program written using Compiled Languages are already compiled, they are much faster than program written in Interpreted Languages.

Creating Java programs

Using an IDE(Integrated Development Environment) such as Eclipse or NetBeans.

Eclipse:

- Execute Eclipse.
- Select file, new, project and enter your project name give your project a meaningful name e.g. myFirstProject. This will create a project which will show on the screen's left column.
- Double click your project name.
  Right click the folder src inside your project.

- Select file, new, class and enter your class/program name give your class a meaningful name e.g. myFirstProgram. This will create your program which will show inside the src folder.
- Run your program. On the command line select Run, or below the command line press the green icon > If you have errors the program will NOT compile until the errors are fixed.

NetBeans:

- Execute NetBeans
- On the top left side of the screen, select File, New Project, Java, Java Application, New, now enter your project name give your project a meaningful name e.g. myFirstProject. This will create a project which will show on the screen's left column. At this time you can leave the information on Project Location, and Project Folder as is, these are the default locations given by NetBeans or you can select different locations. Just remember where you are saving your projects so that you can copy or email your programs. Press Finish and now NetBeans will create what is called a template of the program where you can start writing your program.
- To save your program you can select File, save, or press the picture of a floppy disk on the top left of the screen.
- To Excecute/run your program press on the top the screen a green arrow >. This will compile and run your program, if you have errors this process will let you know so that you can fix them, save and re-run your program.
- For detailed instructions on NetBeans, please refer to its Help section.

Using command line

With editors such as EditPro, Notepad, gedit.

- Goto to the command line of your operating system (Windows, etc.)
  Windows: Start, run, cmd, enter
  Linux Ubuntu: Applications, Accessories, Terminal
- Open the editor of your choice
- Create/Save your program
- Compile your program as follows:
  javac myFirstProgram.java
  This command (javac) will create the compiled program called :
  myFirstProgram.class
  If you have errors the program will NOT compile until the errors are fixed
- Run your compiled program:
  java myFirstProgram.class

Note: Your program's source code file (your typed program) needs to have three parts: the program name, one period, and the word java. I call it firtName.lastName e.g. myFirstProgram.java

What is compiling in Java

When you compile a Java program its compiler (javac) translates your program (the source code) into instructions that the JVM (Java Virtual Machine) can understand. These instructions are known as byte codes and are created in a file that has the same name as your program name and adds .class to it.

```
e.g. : program name  = myFirstProgram.java
       compiled name = myFirstProgram.class
```

Java is an interpreted language not a compiled language. The JVM (Java Virtual Machine) in your computer will interpret and execute the byte codes in your myFirstProgram.class program. The advantage of this technology is that most Operating Systems (Windows, Linux, etc) have their own JVM, capable of executing any Java source code, therefore you can write your program on a Windows XP 32 bit computer and re-compile it and run it in a totally different type of computer using different Operating Systems like Solaris, Linux, etc.

If you create your Java program in a Windows computer and want to run it in a Linux computer, you need to transfer your Java source code from the Windows to the Linux computer, and recompile it. This will create a new .class file. Class files compiled in different operating systems are not compatible.

//end Chapter One : The Basics

Chapter Two : Getting Started

Let's start programming

Every Java program is a CLASS, therefore java has class, it is a classy language.

Documenting a Program

Java provides us with three ways to document our work, they are called REMARKS. Any documentation with REMARKS is ignored by the Java compiler, and is not part of the program instructions.

The symbols /* begin a documentation area, and ends it with */ Such a section can be of any size, even thousands of pages. e.g.:

```
/*
  Author  : Michael Robinson
  Program : myFirstProgram.java
  Purpose : To present, the java class, the main method,
            one variable of data type int
            one variable of data type String
            and the three print command of Java
  Updated : November 12th, 2011 by Michael
*/
```

The symbols // can be placed in any line of code, anything written after them is a remark, therefore ignored by the compiler.

```
        //Create the variable myFirstName and assign my name to it.
        String myFirstName = "Michael"; //This is a String Type variable
```

There is a third documentation procedure called Javadoc which generates html documentation, by adding Javadoc comments in the program code. These are some of the basic Javadoc comments. To see detail Javadoc documentation please see Apendix II.

Javadoc Comments

The symbols **/** .... */** are used by Javadoc. By placing Javadoc comments just before declaration statements, we can add descriptions to classes, methods, variables, and other tools that we have not seen yet such as constructors, and interfaces, e.g.:

```
/** Creating javadocExample class */
public class javadocExample
{
   /** declaring the variable studentID */
   public int studentID;


   /** declaring the main method */
   public static void main(String arg[])
   {
      // call all the methods
   }
}
```

To produce the html javadoc report see Appendix II.

Every program must be documented fully and in detail, when you update a program that you wrote a month ago, or that was written by someone else, if it is not clearly documented, it is impossible to create proper updates.

What are imports

Java is a very large programming language, when loaded it contains basic utilities that allows us to create a lot of programs, however to create some programs we need to import utilities into Java at the beginning of the program prior to the class header (see next section), e.g. if we want to obtain data from the keyboard we need to "import" a java utility called Scanner as follows:

```
import java.util.Scanner;
```

If you include the line

```
import java.util.*;
```

Java will load all the commands (classes) in java.util

What is a Class

In Java all programs are classes, to me a class is equivalent to the perimeter of my house, it has walls, a floor, and a ceiling. Whether it is one bedroom apartment or a 20 room mansion it has a perimeter. Java programs are of all sizes, and their perimeter is the outside class. A program can be of the size of the Windows operating system

(over a million lines of code) or as small as the program that controls the time and temperature of your microwave.

Creating a class

```
public class nameOfProgram
{

}
```

The first line is called "the header", here the header is:

```
public class nameOfProgram
```

public specifies the type of access, *public* means that this program can be accessed by any other program.

class means that this is a *class,* not a method.

nameOfProgram is the name of this program.

When choosing a name make it meaningful. e.g., customerName, accountNumber, customerAddress are better than a, b, c.

My preferred naming convention is called camelCase, the first letter of the name is in lowercase, the first letter of the following words are upper case, and the name is as clear as possible to tell me what it means e.g., myFirstProgram, customerFirstName, customerLastName, accountNumber.

The class MUST have two curly braces { }, the opening brace { and the closing brace }. The rest of the program MUST be inside these two braces.

When you save your program to secondary storage (disk, usb, etc) the program name MUST have the same name as the class name plus one period and the word java, therefore the program name for this class is: nameOfProgram.java

What is a method

A method is a section of code inside a program (class) which contains instructions written by you. Methods always go inside a class.

Methods are independent logical units of programming instructions, and can be re-used as often as necessary. If we have an error in a method, it is restricted in that method which makes finding and fixing errors easier.

Methods can not see each other, however they can communicate among themselves by making calls capable of passing and/or receiving data to/from each other.

Creating a method

Every Java program needs a point to start, so when you start or run a program, Java needs to know where to go, in Java this start method is called main, it is the point of entry. This is what the main method looks like

```
public static void main( String arg[] )
{
    instructions go here
}
```

public means that the method is visible and can be called from other methods.

static means that the method is associated with the class, and is not a specific instance of that class.

void means that the method is not returning any data to the method that called it. If the method instead of void had float, int, String, etc, it is returning data of that type to the calling method.

main is the name of the method.

(String arg[]) always goes next to main, I will explain the meaning of this when we get to *arrays*.

All methods MUST have two curly braces **{ }**, the opening brace { and the closing brace }. All instructions MUST be inside these two braces.

The main method MUST go inside the program class, as follows :

```
public class nameOfProgram
{
    public static void main(String arg[])
    {
       instructions go here
    }
}
```

It can only be ONE main method in a class and it can NOT be inside another method. It can go anywhere but I recommend that it be located as the last method in the class, and of course inside the class.

Most programs need to display messages on a monitor/screen or printer to communicate with the user, even microwaves do that. Java uses multiple commands for this purpose. For the moment I will introduce the “println” command so that we can start programming, later in the “Print Commands” section we will learn in detail how to implement the print, println, and printf commands.

Our first assignment consists in telling the world who we are, using a Java program. We already know what is a class and a main method, inside the main method we are going to implement our first commands as follows:

```
/*
  Author  : Michael Robinson
  Program : nameOfProgram.java
  Purpose : To present, the java class, the main method,
            and display my name using
            the Java println command
  Updated : January 14, 2012
*/

public class nameOfProgram
{
    public static void main(String arg[])
    {
       System.out.println( “Hello World, my name is Joe” );
       System.out.println( “Have a nice day.” );
    }
}
```

Lets break these lines of code:

```
        System.out.println( “Hello World, my name is Joe” );
        System.out.println( “Have a nice day.” );
```

System.out.println means display whatever is between the () and inside the “”, then include a “line feed” or “enter” at the end of the text, so in this case we will see:

```
Hello World, my name is Joe
Have a nice day.
```

Every line of code containg a complete command (sometimes we need multiple lines to give a command to the computer) MUST end with a semicolon ; always.

In the following sections we will be using this println command.

Indentation

Just like a book, programming must follow a structure that allows all persons working in any program to visually understand the sections in it. If every line in a book was left justified, and there were no headings it will be very difficult to read. The same happens in programming.

The class header and its two braces are left justify. All methods inside the class MUST be indented the same amount of spaces, 3, 4, 5 or whatever is best for you. All methods MUST be equally separated by 2 or 3 spaces.

Print Commands

Java uses three distinct commands to print.
System.out.print, System.out.println, and System.out.printf

System.*out*.print

Remember that all statements in Java MUST end with a ;

When a print statement is too long to fit in one line we can split it at the + symbol.

```
System.out.print( "My name is : " +
                  myFirstName +
                  " My ID is : " + myID + "\n" );
```

System.*out*.print displays whatever is between the () following these rules:

whatever is between “” will be printed
the + adds the contents of the next variable
the + adds what is between the “”
the + adds the contents of the next variable
the + adds the enter key which is \n in Java.

System.*out*.println

```
System.out.println( "My name is : " +
                    myFirstName +
                    " My ID is : " + myID );
```

System.*out*.println is exactly the same as System.*out*.print, except that the command \n does not need to be added inside the (), unless

you want an additional line print to be executed, because \n or line print is included in the System.*out*.println with the letters ln meaning add one line feed or carriage return.

The \n command can be inserted at any location in the print, and as many times as needed, as long as it is inside “”, creating an additional line feed or carriage return.

System.*out*.printf

This is the Java implementation of the printf command in C, it is very powerful, and can do all, and more than the print and println commands above.

When a printf statement is too long to fit in one line we can split it at the “, location.

```
System.out.printf( "My name is : %s My ID is : %d\n",
                   myFirstName, myID );
```

printf contains several distinct placeholders where the corresponding data variables will display. These placeholders must be inside “”.

```
    %d = displays variables of int data type
    %s = displays variables of String data type
    %c = displays variables of char data type
    %f = displays variables of float or double data type
         to these data types (float and double) we can request the
         amount of decimals we want, e.g.
         for a float with 2 decimal we use:
  %.2f = displays variables of float data type with 2 decimals.
         We can have as many decimal places as needed.
```

After the “” inside the printf statement we place a , (comma) and then the corresponding variables in the same sequence as the placeholders. e.g.:

```
System.out.printf( "My name is : %s My ID is : %d\n",
                   myFirstName, myID );
```

myFirstName uses the first placeholder %s

myID uses the second placeholder %d

//end **Chapter Two : Getting Started**

Chapter Three : Basic Tools

Numerical Systems

Throughout the existence of humans, we have invented many numeric systems. Here we are going to discuss three types that we will use in programming.

Decimal Numerical System. We use this numerical system everyday. It contains 10 digits from 0 to 9 and it is known as base-10 numerical system because it has 10 numbers. As we know, all our math computations are done with these 10 digits. e.g.:

```
idNumber = 9876;

x = 5 + 2;

addNumbers( 3, 7, 8, 123, 45 );
```

Binary Numerical System. Computers and all digital systems, such as cell phones, digital cameras, etc. use the binary systems or base-2 numeric system, which has 2 digits, 0 and 1. At first, it is hard to understand, but everything a digital system does, like video, sound, graphics, calculations, etc, is represented in binary. Because we use programming languages we write the programs in plain English commands, and the programming language, in this case Java, automatically translate our work into binary, which is what digital systems understand, even the decimal system is translated into binary.

Why binary?
The language of digital equipment is the binary numerical system. Any number can be represented by any sequence of bits (binary digits), which in turn may be represented by any mechanism capable of being in two mutually exclusive states, such as electric or magnetic, on/off.

The first known description of a binary numeral system was presented during the 5th–2nd centuries BC by the Indian scholar Pingala, since then variations have been presented in China, Africa, France, Germany, and the United States. In 1937, Claude Shannon, implemented Boolean algebra and binary arithmetic using electronic relays and switches for the first time in history, in his master's thesis at MIT entitled A Symbolic Analysis of Rela and Switching Circuits,

Shannon's thesis essentially founded practical digital circuit design.

A byte is a sequence of bits (binary digits), today we use 4, 8, 16, 32, 64 bit bytes.

This is an 8 bit byte:

Decimal		Binary
0	=	0 0 0 0 0 0 0 0
1	=	0 0 0 0 0 0 0 1
2	=	0 0 0 0 0 0 1 0
3	=	0 0 0 0 0 0 1 1
4	=	0 0 0 0 0 1 0 0
5	=	0 0 0 0 0 1 0 1
6	=	0 0 0 0 0 1 1 0
7	=	0 0 0 0 0 1 1 1
8	=	0 0 0 0 1 0 0 0
9	=	0 0 0 0 1 0 0 1
10	=	0 0 0 0 1 0 1 0
11	=	0 0 0 0 1 0 1 1
12	=	0 0 0 0 1 1 0 0
13	=	0 0 0 0 1 1 0 1
14	=	0 0 0 0 1 1 1 0
15	=	0 0 0 0 1 1 1 1
16	=	0 0 0 1 0 0 0 0
:		
32	=	0 0 1 0 0 0 0 0
:		
64	=	0 1 0 0 0 0 0 0
:		
126	=	1 0 0 0 0 0 0 0
:		
255	=	1 1 1 1 1 1 1 1

As we can see we start at the right most location of the byte, every bit to the left is double its value. The highest decimal value of an 8 bit byte is 255, which is the highest decimal representation in the ASCII values (please see **ASCII Codes Table** below)

At the present time, to get a stable magnetic state, each bit requires 1,000,000 atoms.

Hexadecimal Numerical System. This numerical system is base-16 it goes for 0 – F, that is from 0–9 then ABCDEF

A = 10

B = 11
C = 12
D = 13
E = 14
F = 15

At one time or another we have experienced computer problems where errors messages show up on the screen such as:

memory dump at: DA9F011231CCABDEDACF765

What this means is that a memory error occurred at memory location DA9F011231CCABDEDACF765.

ASCII Codes Table

ASCII means **American Standard Code for Information Interchange.**

Every character that a computer can make is assigned a numerical value, the ASCII code is limited to the first 128 characters/numbers, from 0 to 127. There are 33 non-printing control characters like esc, bell, etc. that affects how text and space is processed; 95 are printable characters such as a, b, c, 1, 2.

The Extended ASCII characters begin at 128 and end at 255, these groups contain non-english alphabetical characters such as:

ASCII		Symbol
128	=	Ç
164	=	ñ
142	=	À
230	=	µ
145	=	æ
228	=	Σ

Work on ASCII formally began on October 6, 1960. The first edition of the standard was published during 1963, a major revision during 1967, and the most recent update during 1986.

The most commonly used character encoding on the World Wide Web was US-ASCII until December 2007.

A program that prints the ASCII codes can be seen at the Miscellaneous Chapter at section “ASCII Codes Program”.

ASCII TABLES

Numbers

KEY NAME	ASCII CODE	KEY NAME	ASCII CODE
1	**49**	**6**	**54**
2	**50**	**7**	**55**
3	**51**	**8**	**56**
4	**52**	**9**	**57**
5	**53**	**0**	**58**

Alphabet

KEY NAME	ASCII CODE	KEY NAME	ASCII CODE
A	**65**	**a**	**97**
B	**66**	**b**	**98**
C	**67**	**c**	**99**
D	**68**	**d**	**100**
E	**69**	**e**	**101**
F	**70**	**f**	**102**
G	**71**	**g**	**103**
H	**72**	**h**	**104**
I	**73**	**i**	**105**
J	**74**	**j**	**106**
K	**75**	**k**	**107**
L	**76**	**l**	**108**
M	**77**	**m**	**109**
N	**78**	**n**	**110**
O	**79**	**o**	**111**

P	**80**	**p**	**112**
Q	**81**	**q**	**113**
R	**82**	**r**	**114**
S	**83**	**s**	**115**
T	**84**	**t**	**116**
U	**85**	**u**	**117**
V	**86**	**v**	**118**
W	**87**	**w**	**119**
X	**88**	**x**	**120**
Y	**89**	**y**	**121**
Z	**90**	**z**	**122**

Some Extended Characters

KEY NAME	ASCII CODE	KEY NAME	ASCII CODE
á	**160**	**ü**	**129**
é	**130**	**ñ**	**164**
í	**161**	**Ñ**	**165**
ó	**162**	**¿**	**168**
ú	**163**	**¡**	**173**

To obtain the extended ASCII characters in Windows, press and hold the ALT key and enter the corresponding ASCII code, for instance to obtain **Ñ** enter **ALT 165**

To write the letter A in binary, we write its ASCII code

65 = 0 1 0 0 0 0 0 1

A program that prints the ASCII codes can be seen at the Miscellaneous Chapter at section “ASCII Codes Program”.

The Unicode Standard.

This is another coding system that can represent over 100,000 characters. The origins of Unicode date back to 1987. The latest version of Unicode consists of more than 109,000 characters covering 93 scripts, a set of code charts for visual reference, display of text containing both right-to-left scripts, such as Hebrew and Arabic, as well as left-to-right scripts. As of 2011, the most recent major revision of Unicode is Unicode 6.0.

All the values represented in the ASCII codes are in the same locations in the Unicode Standard.

Data Types

In math we assign numbers to variables such as a, b, c, etc. e.g.

```
a = 5      where a becomes 5
b = 12.4   where b becomes 12.4
c = a + b  where c becomes 17.4
```

In computer Science a variable is a descriptive name given to a known amount of information, so that we can use such a variable regardless of the value it represents. The value of variables can change during the life of the program.

In Java we do the same, however before assigning a value to a variable, we assign what we call a data type to the variable.

The *data types* for the above numbers are called *primitive* data types.

```
Primitive   variable       value
  int          a      =       5;
  float        b      =    12.4;
  float        c      =   a + b;
```

Primitive Data Types

Each Primitive has its own name, occupying a specific amount of Bytes, and containing a minimum and a maximum value. The amount of Bytes and values depend on the computer the program is running; large computer systems have different values, the following are the typical values on today's standard personal computers.

Today's Primitive Data Types use the Unicode Standard, all the values represented in the ASCII codes are in the same locations in the Unicode Standard.

unsigned = uns

Name	Bits	Min Val	Max Val
byte	8	-128	127
char	8	-32,768	32,767
uns-char	8	0	65,535
int	32	-2,147,483,648	2,147,483,64
short	16	-32,768	32,767
uns-short	16	0	65,535
long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	32	-+1.701411E-38	-+1.701411E38
double	64	-+1.0E-307	-+1.0E307
boolean	1	true/false	

The following describes what each Primitive Data Type means:

byte
An eight bit byte allows us to write data values from -128 to 128 or a maximum positive number of 255 by turning all 8 bits on, and then adding them.

Bit number	8	7	6	5	4	3	2	1
State on	1	1	1	1	1	1	1	1
Decimal value	128	64	32	16	8	4	2	1 = 255

char
The char data type is used to place a single character, such as a letter, number, or symbol. Numbers are treated as characters not numbers.

int
The int will hold a whole number. If you placed a number with decimals into an int, the decimals will be dropped.

Short
A short will also hold a whole number but of less value than an int, see table above.

long
A long will also hold a whole number but of greater value than an int, see table above.

float
A float holds whole numbers with decimals, for ranges see table

above.

double
A double holds whole numbers with decimals, with much greater values than the float, for ranges see table above.

Boolean
The boolean type has only two values, true or false. When we declare a variable of type boolean we can only assign it to be either true or false, e.g.

```
boolean electricCurrent = true;
boolean electricCurrent = false;
```

To see a more detailed description of the Java Primitive Data Type please see:
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

What can we do with Primitives

Since primitives are used to declare numbers then any number we use needs to have a primitive.

```
int amountOfCreditsTaken = 527;
int maxCreditsAllowedPerSemester = 18;
int semestersPerYear = 3;
```

So if we want to find out how many years it took to obtain the above amountOfCreditsTaken we can do the following:

```
int crYear = ( amountOfCreditsTaken / maxCreditsAllowedPerSemester );
int years  = ( crYear / semestersPerYear );

System.out.println( “Amount of years = “ + years );
```

In this previous line of code we see + years after the ending “, this means to add the value of years to the data between the “”, so the output is:

```
Amount of years = 9
```

The Primitive Data Type int named years, contains the answer.

As you can see assigning clear names to variables, makes life very easy when programming.

Much better than

```
int a = 527;
int b =  18;
int c =   3;
int y = ( (a / b) / c );
```

You probably noticed a big problem we can get into when we choose the wrong type of primitive.

Notice that 527 / 18 = 29.27777777
and 29.27777777 / 3 = 9.7592592593

But when we do the computations using int instead of float we get 9 not 9.7592592593 which is the correct answer.

Reason: when we divide one integer by another integer and assign the result to an integer, Java “truncates” all results, meaning it ignores all decimals, so to avoid this problem, we need to select the correct data types or use a Java tool called “casting”.

To cast a variable means to convert or promote its primitive data type to a higher primitive. int to float, int to double, float to double, etc.

When we divide an int by another int we must cast either one of the ints as follows:

instead of:

```
int crYear = ( amountOfCreditsTaken / maxCreditsAllowedPerSemester );
```

we make crYear into a float to accept the computation, and using the (float) command we cast either of the other two ints by placing (float) on front of either variables as follows:

```
float crYear = ( (float) amountOfCreditsTaken /
                 maxCreditsAllowedPerSemester );
```

or

```
float crYear = ( amountOfCreditsTaken /
                 (float) maxCreditsAllowedPerSemester );
```

now we make years into a float

```
float years  = ( crYear / semestersPerYear );
```

```
System.out.println( “Amount of years = “ + years );
```

Since crYear is a float there is not need to cast it. When we do a mathematical operation between numbers of different primitive types the result is of the larger Primitive Data Type.

Sample of other Primitive variables and values:

This is called: creating a variable and assigning a value to it.

```
double speedOfLight   = 186,282;    //miles per second
float  earthDiameter  = 7,926.41; //miles
int    myAge          = 105;
float  myHeigth       = 5.9;
```

We can declare a variable without assigning a value to it, however I recommend that you become used to assigning a value, usually zero, because some languages like ANSI C, if you do not assign a value to a new variable, it assigns the value found at the memory location where the variable gets placed, which can cause big problems.

Numerical Operators

There are some special symbols called Operators that perform specific operations on one, two or three numbers, giving us a result.

```
In math we say:      x = 3 + 5
but we never say:    x = x + 5
```

In computer science we can say both of the above, the second method needs explanation:

```
int basket = 5; //creates variable basket and assign value 5 to it.

    basket = basket + 1; //add one to the current value of basket
                         //the new value of basket is 6

    basket = basket - 3; //take 3 from to the current value of basket
                         //the new value of basket is 3
```

In the same manner we can implement multiplication and division.

```
    basket = basket * 3; //NOTE * means multiplication

    basket = basket / 3; //NOTE / means division
```

The following section shows another way to use the math operators:

--, ++, -=, +=, *=, /=

```
int counter = 0; //create and initialize counter to 0

counter++;       //adds 1 to counter, = 1

counter *= 6;    //multiply counter by 6, = 6

counter /= 2;    //divide counter by 2, = 3

counter--;       //take one from counter, = 2

counter -= 2;    //take 2 from counter, = 0

counter += 100;  //add 100 to counter, = 100
```

Other Operators

Variations of -- and ++ before and after the variable.
if we have:

```
int counter = 100;
System.out.println( "counter = " + counter);
```

prints: counter = 100

and

```
System.out.println( "counter = " + counter++);
```

also prints : counter = 100

but the value of counter after the printing is 101, because counter++ means to add 1 to counter after it prints (in this case).

but

```
System.out.println( "counter = " ++counter );
```

will print : counter = 101

because ++counter first adds 1 to the counter value, then prints.
The same happens to the -- command before and after the variable.

Operators > and <

The operator > means left value is greater than right value e.g.:

```
5 > 2  same as in math
```

The operator < means left value is smaller than right value e.g.:

```
2 < 5  same as in math
```

Operators >= and =<

The operator >= means left value is greater or equal
to right value e.g.:

```
5 >= 5
5 >= 4
```

The operator <= means left value is smaller or equal
to right value e.g.:

```
5 <= 5
4 <= 5
```

Operators =, ==, !=

This is where many people get confused and cause programming errors.
The operator = means the right value is assign to the left value e.g.

```
total = 5; //5 is assigned to the variable total
```

The operator == means equivalence, we check to see if the left value
is equivalent to the right value

```
if( total == 5 ) //if total is equivalent to 5,
                 //does total and 5 have the same value?
```

The operator != mean not equal, we check to see if the left value
is not equal to the right value

```
if( total != 5 ) //if total is not equal to 5
```

Operators && and ||

The operator && means logical “and”

```
if( cold && raining ) //if cold and raining
```

The operator || means logical “or”

```
if( cold || raining ) //if cold or raining
```

Operator %

The % operator also known as modulus and mod, means remainder when dividing two numbers.

It looks like division but it is NOT

```
5 % 2 = 1
```

means that 2 exists twice in 5 leaving 1 as the the modulus

```
182 % 3 = 2
```

3 exists 60 times in 182 leaving 2 as the modulus

```
55 % 7 = 6
```

7 exists 7 times in 55 leaving 6 as the modulus

```
2000 % 4 = 0
```

4 exist 500 times in 2000 leaving 0 as the modulus

The % operator is very useful in programming, allowing us to perform very complex operations easily.

Math using primitives

Java provide us with already made methods to do many math computations. The math class is called Math.

For Detailed description of the Math methods see Appendix I.

The following program shows the implementation of some Math methods.

```
public static void processAbsoluteValues(int i, int j, double x, double y )
{
      // Math.abs(i)
      // The absolute value of a number is equal to
      // the number if the number is positive or
      // zero and equal to the negative of the number
      // if the number is negative.
      System.out.println("|" + i + "| is " + Math.abs(i));
```

```
      System.out.println("|" + j + "| is " + Math.abs(j));
      System.out.println("|" + x + "| is " + Math.abs(x));
      System.out.println("|" + y + "| is " + Math.abs(y));

}//end processAbsoluteValues(int i, int j, double x, double y )


public static void processRoundValues( double x, double y )
{
      // Math.round(x)
      // Truncating and Rounding functions
      // You can round off a floating point number
      // to the nearest integer with round()
      System.out.println(x + " is approximately " + Math.round(x));
      System.out.println(y + " is approximately " + Math.round(y));

}//end processRoundValues( double x, double y );


public static void processCeilingValues( double x, double y )
{
      // Math.ceil(x)
      // The "ceiling" of a number is the smallest integer greater than or
      // equal to the number. Every integer is its own ceiling.
      System.out.println("The ceiling of " + x + " is " + Math.ceil(x));
      System.out.println("The ceiling of " + y + " is " + Math.ceil(y));

}//end processCeilingValues( double x, double y );


public static void processFlooringValues( double x, double y )
{
       // Math.floor(i)
       // The "floor" of a number is the largest integer less than
       // or equal to the number. Every integer is its own floor.
       System.out.println("The floor of " + i + " is " + Math.floor(i));
       System.out.println("The floor of " + j + " is " + Math.floor(j));
       System.out.println("The floor of " + x + " is " + Math.floor(x));
       System.out.println("The floor of " + y + " is " + Math.floor(y));

}//end processFlooringValues( double x, double y );


public static void processMinimunValues(int i, int j, double x, double y )
{
       // min() returns the smaller of the two arguments you pass it
       System.out.println("min(" + i + "," + j + ") is " + Math.min(i,j));
       System.out.println("min(" + x + "," + y + ") is " + Math.min(x,y));
       System.out.println("min(" + i + "," + x + ") is " + Math.min(i,x));
       System.out.println("min(" + y + "," + j + ") is " + Math.min(y,j));

}//end processMinimunValues(int i, int j, double x, double y )


public static void processMaximunValues(int i, int j, double x, double y )
{
```

```
        // There's a corresponding max() method
        // that returns the larger of two numbers
        System.out.println("max(" + i + "," + j + ") is " + Math.max(i,j));
        System.out.println("max(" + x + "," + y + ") is " + Math.max(x,y));
        System.out.println("max(" + i + "," + x + ") is " + Math.max(i,x));
        System.out.println("max(" + y + "," + j + ") is " + Math.max(y,j));

 }//end processMaximunValues(int i, int j, double x, double y )


public static void processTrigFunctionsValues(int i, int j, double x, double y)
{
       // The Math library defines a couple of useful constants:
       System.out.println("Pi is " + Math.PI);
       System.out.println("e is " + Math.E);

       // Trigonometric methods
       // All arguments are given in radians
       // Convert a 45 degree angle to radians
       double angle = 45.0 * 2.0 * Math.PI/360.0;
       System.out.println("cos(" + angle + ") is " + Math.cos(angle));
       System.out.println("sin(" + angle + ") is " + Math.sin(angle));

       // Inverse Trigonometric methods
       // All values are returned as radians
       double value = 0.707;
       System.out.println("acos(" + value + ") is " + Math.acos(value));
       System.out.println("asin(" + value + ") is " + Math.asin(value));
       System.out.println("atan(" + value + ") is " + Math.atan(value));

 }//end processTrigFunctionsValues(int i, int j, double x, double y );


 public static void processExponentialValues()
 {
       // Exponential and Logarithmic Methods
       // exp(a) returns e (2.71828...) raised
       // to the power of a.
       System.out.println("exp(1.0) is "  + Math.exp(1.0));
       System.out.println("exp(10.0) is " + Math.exp(10.0));
       System.out.println("exp(0.0) is "  +  Math.exp(0.0));

 }//end processExponentialValues();


 public static void processLogValues()
 {
       // log(a) returns  the natural
       // logarithm (base e) of a.
       System.out.println("log(1.0) is "    + Math.log(1.0));
       System.out.println("log(10.0) is "   + Math.log(10.0));
       System.out.println("log(Math.E) is " + Math.log(Math.E));

 }//end processLogValues();
```

```
public static void processPowerValues()
{
      // pow(x, y) returns the x raised to the yth power.
      System.out.println("pow(2.0, 2.0) is "  + Math.pow(2.0,2.0));
      System.out.println("pow(10.0, 3.5) is " + Math.pow(10.0,3.5));
      System.out.println("pow(8, -1) is "     + Math.pow(8,-1));

}//end processPowerValues();


public static void processSquareRootsValues()
{
      // sqrt(i) returns the square root of i.
      int i = 0;
      for (i=-7; i < 10; i++)
      {
        System.out.println("The square root of " + i + " is " + Math.sqrt(i));
      }

}//end processSquareRootsValues()


public static void processRandomValues()
{
      // Finally there's one Random method
      // that returns a pseudo-random number
      // between 0.0 and 1.0;

      System.out.println("Here's one random number: " + Math.random());
      System.out.println("Here's another random number: " + Math.random());

}//end public static void processRandomValues()



public static void main(String[] args)
{
    int i = 7;
    int j = 9;
    double x = 72.5;
    double y = 0.34;

    processAbsoluteValues(i, j, x, y );
    processRoundValues( x, y );
    processCeilingValues( x, y );
    processFlooringValues( x, y );
    processMinimunValues(i, j, x, y );
    processMaximunValues(i, j, x, y );
    processTrigFunctionsValues(i, j, x, y );
    processExponentialValues();
    processLogValues();
    processPowerValues();
    processSquareRootsValues();
    processRandomValues();

    System.out.println("\nEnd of Program");
```

```
    }//end public static void main(String[] args)

}//end public class mathExamples
```

Wrappers

As we learned in the Primitive Data Types section, Primitives do not have methods, however sometimes we need to perform functions on numbers and we need methods, to do this Java has given us the Wrapper Classes. For the eight Primitive Data types we have eight Wrapper Classes.

Primitive	**Wrapper**
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Notice that the first letter of the Primitives is always lower case, and the Wrappers first letter is always upper case. The first letter of all classes is always upper case.

String Class Data Types

To work with numbers we use the primitive data types, to work with characters such as “aec$ ^%FtaA01982.GA/}]” we use the String class.

The String Java class has over 1,000 methods. This class is used for string variables containing a sequence of characters from 0 to 2gig in length, the characters in it can be the combination of letters, numbers and symbols, e.g.

```
String myName = “ Albert# &(723,.$ Einstein\? “;
```

Some of the differences between primitives and Strings are:

- Strings accept any character. e.i:
  ```
  String address = “123 main street, apt #12 N.Y.”;
  ```
- Primitives accept only numbers, except for the char data type that accepts one character inside to single quotes e.i.:
  ```
  char letter = 'a';
  ```
  and the boolen data type that accept true or false e.i.:
  ```
  boolean valid = true;
  boolean open  = false;
  ```

However the main difference between Primitives and Strings is that the String class has methods and Primitives do not.

Because of its methods Strings allows us to do just about anything, if a string contains numbers, using String methods, we can convert the String numbers into primitive numbers and do math with them.

Methods inside classes are logical sections of code that do many things for us, they are programs already made that we can apply to our Strings. For instance if I create a String called learning:

```
String learning = “I am learning Java”;
```

and I want to convert it to all capital letters, all I need to do is ask the “learning string” to allow me to use its method that does this for me, I do not have to create a program to do it, this is how I would do it.

```
String learning = "I am learning Java";
System.out.println( learning );

//convert learning toUppercase
learning = learning.toUpperCase();
System.out.println( learning );
```

Now to convert it to lower case:

```
//convert learning.tolowercase
learning = learning.toLowerCase();
System.out.println( learning );
```

String has over 1,000 already made methods that we can use.

For instance when we are comparing two strings we can use the compareTo method in String:

```
data[x-1].compareTo( data[x] ) == 0   means equal
data[x-1].compareTo( data[x] ) != 0   means not equal
data[x-1].compareTo( data[x] ) < 0    first one less than second one
data[x-1].compareTo( data[x] ) > 0    first one greater than second
```

First String Program

Let's write our first Java program. We need to create two variables, one to store my ID number of data type int, called myID, and another to store my name of data type String, called myName.

```
/*
```

```
  Author  : Michael Robinson
  Program : myFirstStringProgram.java
  Purpose : To present, the java class, the main method,
            one variable of data type int
            one variable of data type String
            and the three print command of Java
  Updated : November 12th, 2011 by Michael
*/

public class myFirstStringProgram
{
    public static void main(String arg[])
    {
       int    myID        = 777; //I am creating variable myID, and
                                 //assign the value 007 to it.

       //Create the variable myFirstName and assign my name to it.
       String firstString =  "I am the first String";

       String myFirstName = "Michael";

       System.out.print("My name is : " + myFirstName +
                        " My ID is : " + myID + "\n");

       System.out.println("My name is : " + myFirstName +
                          " My ID is : " + myID );

       System.out.printf("My name is : %s My ID is : %d\n",
                          myFirstName, myID );

    }//end public static void main(String arg[])
}//end public class myFirstStringProgram
```

If you are using an IDE or the command line method to write your program, follow the instructions on the corresponding sections.

Your output (result) will look like this:

```
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
```

The above program contains the following sections:
Documentation section, identifies the program.
Class section, the program's perimeter called myFirstStringProgram.
Main method section, contains instructions for Java to do.

In the main method we do the following:
 Create and initialize the variables myID and myName.
 Print the values in the above variables, using three distinct ways
 of printing in java.

String Tokenizer

When analyzing data, lets say the contents of a book, we need to be able to split the entire book into words. The String Tokenizer is what we use to do so. This is a simple but clear program that does just that.

```
/**
Author  : Michael Robinson
Program : tokenizer.java
Purpose : To present String tokenizer class
          and some of its methods

Updated : April 26, 2011
*/

public class tokenizer
{
   public static void split1()
   {
      System.out.println("\n\nSpliting 'one two three four' delimited by ' '");
      // Create a string to tokenize.
      String str = "one two three four";
      System.out.println("\t  " + str);

      // Get the tokens, using a space delimiter.
      String[] tokens = str.split(" ");

      // Display the tokens.
      for (String s : tokens)
      {
         System.out.println("\t  " + s);
      }
   }


   public static void split2()
   {
      System.out.println("\n\nSpliting 'one and two and three and four' " +
                         "delimited by ' and '");
      // Create a string to tokenize.
      String str = "one and two and three and four";
      System.out.println("\t  " + str);

      // Get the tokens, using " and " as the delimiter.
      String[] tokens = str.split(" and ");

      // Display the tokens.
```

```
      for (String s : tokens)
      {
         System.out.println("\t  " + s);
      }
   }


   public static void split3()
   {
      System.out.println("\n\nSpliting 'joe@gaddisbooks.com' delimited by '@.'");
      // Create a string to tokenize.
      String str = "joe@gaddisbooks.com";
      System.out.println("\t  " + str);

      // Get the tokens, using @ and . as delimiters.
      String[] tokens = str.split("[@.]");

      // Display the tokens.
      for (String s : tokens)
      {
         System.out.println("\t  " + s);
      }
   }


   public static void split4()
   {
      System.out.println("\n\nSpliting 'Jill$Billy%Becky*Tara&Mary' delimited " +
                         "by '[$%*&]'");

      // Create a string to tokenize.
      String str = "Jill$Billy%Becky*Tara&Mary";
      System.out.println("\t  " + str);

      // Get the tokens, using a space delimiter.
      String[] tokens = str.split("[$%*&]");

      // Display the tokens.
      for (String s : tokens)
      {
         System.out.println("\t  " + s);
      }
   }


   public static void main(String[] args)
   {
      split1();
      split2();
      split3();
      split4();
      System.out.println("End of program");
   }

}//end public class tokenizer
```

```
This is the output:

Splitting 'one two three four' delimited by ' '
       one two three four
       one
       two
       three
       four


Splitting 'one and two and three and four' delimited by ' and '
       one and two and three and four
       one
       two
       three
       four


Splitting 'joe@gaddisbooks.com' delimited by '[@.]'
       joe@gaddisbooks.com
       joe
       gaddisbooks
       com


Splitting 'Jill$Billy%Becky*Tara&Mary' delimited by '[$%*&]'
       Jill$Billy%Becky*Tara&Mary
       Jill
       Billy
       Becky
       Tara
       Mary
End of program
```

//end **Chapter Three : Basic Tools**

Chapter Four : Building a house

Calling Methods

In programming it is very important to be organized, good use of methods is the best way to achieve organization in Java.

In Java we can have as many methods as we want. The main method is the entry point of Java. I use it to create variables and as a the central control point from where I call all methods, we can also call methods from inside another methods.

To show how to use methods I say that Java has three types of methods with variations.

Let's assume we have four methods called printMyInfo1, printMyInfo2, printMyInfo3, and printMyInfo4.

Methods that do not pass any data

```
  //calls method NOT passing any data to printMyInfo1
  printMyInfo1();
```

The program calling the printMyInfo1() method looks like this:

```
/*
  Author  : Michael Robinson
  Program : mySecondProgram.java
  Purpose : To present, the java class, the main method,
            calling printMyInfo1() method
  Updated : November 12th, 2011 by Michael
*/

public class mySecondProgram
{
    public static void printMyInfo1()
    {
       System.out.print("Hi I am at the printMyInfo1() method\n");

    }//end public static void printMyInfo1()


    public static void main(String arg[])
    {
       //calls method NOT passing any data to printMyInfo1
       printMyInfo1();
```

```
        System.out.print("Hi I am back at main from printMyInfo1()"+
                         " method\n");

    }//end public static void main(String arg[])

}//end public class myFirstStringProgram
```

This is the output:

```
Hi I am at the printMyInfo1() method
Hi I am back at main from printMyInfo1() method
```

A variable or a method that is declared public is publicly accessible by any member of the project. Any class or method can freely access other public methods and variables of another class.

Methods that pass data

```
  //calls method passing myName and myID to printMyInfo2
  printMyInfo2( myName, myID );
```

The program calling the printMyInfo2() method looks like this:

```
/*
  Author   : Michael Robinson
  Program : mySecondProgram.java
  Purpose : To present, the java class, the main method,
            one variable of data type int
            one variable of data type String
            and the three print command of Java
            Calls four methods
  Updated : November 12th, 2011 by Michael
*/

public class mySecondProgram
{
    public static void printMyInfo2( String myFirstName, int myID )
    {
       System.out.print("\nHi I am at printMyInfo2( String
                         myFirstName, int myID ) method\n");

       System.out.print( "My name is : " + myFirstName +
                        " My ID is : " + myID + "\n" );
```

```
        System.out.println( "My name is : " + myFirstName +
                            " My ID is : " + myID );

        System.out.printf( "My name is : %s My ID is : %d\n",
                           myFirstName, myID );

    }//end public static void printMyInfo2


    public static void main(String arg[])
    {
        int    myID        = 777; //I am creating variable myID, and
                                  //assigning value 777 to it.

        //Create the variable myFirstName and assign my name to it.
        String myFirstName = "Michael";

        //Methods that pass data
        //calls method passing myName and myID to printMyInfo2
        printMyInfo2( myFirstName, myID );
        System.out.print("Hi I am back at main from public static" +
                         " void printMyInfo2( String myFirstName," +
                         " int myID ) method\n");

    }//end public static void main(String arg[])

}//end public class myFirstStringProgram
```

This is the output:

```
Hi I am at the public static void printMyInfo2( String myFirstName,
int myID ) method
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
Hi I am back at main from public static void printMyInfo2( String
myFirstName, int myID ) method
```

A variable or a method that is declared public is publicly accessible by any member of the project. Any class or method can freely access other public methods and variables of another class.

Methods that return data to calling statement which called a method either passing or not passing data.

Here we have the third manner of calling a method, and I am showing a

variation, also the two previous examples are included.

```
  //calls method passing myName and myID to printMyInfo3
  //and expects data to be returned to the int result variable
  int result =  printMyInfo3( myName, myID );

  //calls method NOT passing data to printMyInfo4
  //and expects data to be returned to the int result variable
  int result =  printMyInfo4();
```

In this example I am declaring and assigning data to two variables, then I am calling four methods as described above.

```
/*
  Author  : Michael Robinson
  Program : mySecondProgram.java
  Purpose : To present, the java class, the main method,
            one variable of data type int
            one variable of data type String
            and the three print command of Java
            Call four methods
  Updated : November 12th, 2011 by Michael
*/

public class mySecondProgram
{
    //calls method NOT passing any data to printMyInfo1
    public static void printMyInfo1()
    {
       System.out.print("Hi I am at the printMyInfo1() method\n");
    }//end public static void printMyInfo1()


    //Methods that pass data
    //calls method passing myName and myID to printMyInfo2
    public static void printMyInfo2( String myFirstName, int myID )
    {
       System.out.print("\nHi I am at printMyInfo2( String
                         myFirstName, int myID ) method\n");

       System.out.print( "My name is : " + myFirstName +
                        " My ID is : " + myID + "\n" );

       System.out.println( "My name is : " + myFirstName +
                           " My ID is : " + myID );
```

```
        System.out.printf( "My name is : %s My ID is : %d\n",
                           myFirstName, myID );

    }//end public static void printMyInfo2
```

Methods that return data

Java also has methods that return data to its calling statement. Note that in this example, in main we are making this call:

```
    int result1 =  printMyInfo3( myFirstName, myID );
```

Also note the first line of the called method:

```
    public static int printMyInfo3( String myFirstName, int myID )
```

Notice that instead of "void" we have "int" which means that this method will return a variable of data type "int".

The last line in this method is:

```
    return( myID * 2 );
```

which means that it is returning the value of (myID * 2) which in this case is an int.

The following are several examples of methods that return data:

```
    //Calls method printMyInfo3 passing myName and myID and
    //expects data to be returned to the int result variable
    public static int printMyInfo3( String myFirstName, int myID )
    {
        System.out.print( "\nHi I am at int printMyInfo3( String
                           myFirstName, int myID ) method\n" );

        System.out.print( "My name is : " + myFirstName +
                          " My ID is : " + myID + "\n" );

        System.out.println( "My name is : " + myFirstName +
                            " My ID is : " + myID );

        System.out.printf( "My name is : %s My ID is : %d\n",
                           myFirstName, myID );

        System.out.print( "now returning myID * 2, which " +
```

```
                    "will be printed at main\n" );

   return( myID * 2 );
}//end public static int printMyInfo3


//calls method NOT passing data to printMyInfo4
//and expects data to be returned to the int result variable
public static int printMyInfo4()
{
   System.out.print("\nHi I am at the public static int " +
                    "printMyInfo4() method\n");

   System.out.print("returning the value 999 which will " +
                    "be printed at main\n");

   return( 999 );
}//end public static int printMyInfo4()


public static void main(String arg[])
{
   int    myID        = 777; //I am creating variable myID, and
                             //assign the value 777 to it.

   //Create the variable myFirstName and assign my name to it.
   String myFirstName = "Michael";

   //calls method NOT passing any data to printMyInfo1
   printMyInfo1();
   System.out.print("Hi I am back at main from printMyInfo1()"+
                    " method\n");
   //Methods that pass data
   //calls method passing myName and myID to printMyInfo2
   printMyInfo2( myFirstName, myID );
   System.out.print("Hi I am back at main from public static" +
                    " void printMyInfo2( String myFirstName," +
                    " int myID ) method\n");

   //Methods that return data to calling statement which called
   //the method either passing or not passing data.
   //calls method passing myName and myID to printMyInfo3
   //and expects data to be returned to the int result variable
   int result1 =  printMyInfo3( myFirstName, myID );
   System.out.print("Hi I am back at main from public static" +
                    " int printMyInfo3( String myFirstName," +
```

```
                        " int myID ) method\n");

        System.out.printf("I received the value %d\n" , result1);

        //calls method NOT passing data to printMyInfo4
        //and expects data to be returned to the int result variable
        int result2 =  printMyInfo4();

        System.out.print("Hi I am back at main from public static" +
                        " int printMyInfo4() method\n");

        System.out.printf("I received the value %d\n" , result2);

    }//end public static void main(String arg[])

}//end public class myFirstStringProgram
```

This is the output:

```
Hi I am at the public static void printMyInfo1() method
Hi I am back at main from public static void printMyInfo1() method

Hi I am at the public static void printMyInfo2( String myFirstName,
int myID ) method
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
Hi I am back at main from public static void printMyInfo2( String
myFirstName, int myID ) method

Hi I am at the public static int printMyInfo3( String myFirstName,
int myID ) method
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
My name is : Michael My ID is : 777
now returning myID * 2, which will be printed at main
Hi I am back at main from public static int printMyInfo3( String
myFirstName, int myID ) method
I received the value 1554

Hi I am at the public static int printMyInfo4() method
returning the value 999 which will be printed at main
Hi I am back at main from public static int printMyInfo4() method
I received the value 999
```

A variable or a method that is declared public is publicly accessible by any member of the project. Any class or method can freely access other public methods and variables of another class.

Pause Program

In programming we need to stop our programs at any place so that we can see what the program is doing, this is called debugging. Java has a class called wait() which allows us to stop a process for a specific amount of time, but it can be interrupted by other processes through a class called threads. However, when debugging we want the program to stop until we tell it to continue, we need total control.

For this purpose I created a program called pauseClass.java which has a method called pause. I use this method all the time to do my debugging, and I think it will be useful to you. So here it is:

Do not worry about understanding this program in detail at this time.

```
/**
Author  : Michael Robinson
Program : pauseClass.java
Purpose : This program creates a method called pause()
          which allows the user to stop the program
          at any particular place.

Updated : April 26, 2011
*/

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;


public class pauseClass
{
   public static void pause() //by Michael Robinson 3-12-2011
   {
      System.out.print("\nPress ENTER to continue ");   // make a prompt

      //open a reader over the keyboard (System.in) stream.
      //class       object                          method
      BufferedReader in = new BufferedReader( new InputStreamReader( System.in ));

      //checking for errors
      try //execute the following line of code
      {
         String line = in.readLine();  // read one line.
      }
      catch (IOException ioe)
      {  //if an error was detected execute the following
         System.out.println("Something went wrong reading IO:");
         ioe.printStackTrace();
```

```
      }

   }//end  public pause()


   public static void main(String arg[])
   {
      pause();

      System.out.println("\n  Thank you!\nend of program");

   }//end public static void main(String arg[])

}//end public class pauseClass
```

Notice that there is a method called:

```
   public static void pause()
```

Do not worry about understanding this program at this time. To stop your program at any place use the pause method as shown the in the main method:

```
         pause();
```

When this method gets called you will receive this message:

```
         Press ENTER to continue
```

To allow the program to continue processing just press the ENTER key. You can place the place() command at any location in you program and as many times as needed.

Variables

Global Variables

In the Methods section we learned that methods do not see each other, however they can communicate with each other by passing and returning data to each other. Java in general has two types of variables, Global and Local. The Global variables are declared/created outside of all methods and they can be declared only once. The advantage of Global variables is that they can be seen by all methods, at the same time the disadvantage is that they can be modified by any method, to solve that problem we can make any variable Global or Local variable into a constant variable so that it can only be read but not changed (see **Private – Final Variables section**).

Private – Final Variables

These type of variables are also called Constant variables because they do not change value once they are created.

The following program shows how to create and use constants.

```
/*
Author  : Michael Robinson
Program : finalStatic.java
Purpose : To present the constant data type
          data type that can not be changed
          once they are declared and a value is assigned

Updated : May 1, 2011
*/


public class finalStatic
{
   public static final double  monthlyRent = 950;
   public static final Integer yearsLeased = 30;


   public static void main(String arg[])
   {
      System.out.printf( "Monthly Rent  %.2f\n", monthlyRent);
      System.out.printf( "Years Leased  %d\n",   yearsLeased);

      System.out.printf( "Total Rent Income :  %.2f \n",
                         ( monthlyRent * yearsLeased * 12 ));
   }

}//end public class finalStatic
```

This is the output:

```
Monthly Rent  950.00
Years Leased  30
Total Rent Income :  342000.00
```

Local Variables

Variables that are created in or passed to a Method, once in that Method is a local variable. What is usually hard to understand is that we can have local variables that have the same name in multiple methods, and they are different. The easiest example to explain this is : We get 200 persons called Maria and we send them to 200 different cities. Each Maria in each city is different. Think of Maria as the name of the variable and each city as a method.

This is an interesting example:

```
/**
Author  : Michael Robinson
Program : variables.java
Purpose : To show the global and local variables

Updated : May 4, 2011
*/

public class variables
{
    static double interestRate = 3.5;

    public static void firstLoan( int loanAmount )
    {
       interestRate = interestRate + .25;
       System.out.printf( "I borrowed %d at %.2f interest.\n", loanAmount,
                                                               interestRate);
    }

    public static void secondLoan( int loanAmount )
    {
       interestRate = interestRate + .25;
       System.out.printf( "I borrowed %d at %.2f interest.\n", loanAmount,
                                                               interestRate );
    }

    public static void thirdLoan( int loanAmount )
    {
       interestRate = interestRate + .25;
       System.out.printf( "I borrowed %d at %.2f interest.\n", loanAmount,
                                                               interestRate );
    }

    public static void fourthLoan( int loanAmount )
    {
       interestRate = interestRate + .25;
       System.out.printf( "I borrowed %d at %.2f interest.\n", loanAmount,
                                                               interestRate );
    }


    public static void main(String arg[])
    {
       firstLoan( 100 );
       secondLoan( 200 );
       thirdLoan( 300 );
       fourthLoan( 400 );

       int loanAmount = 500;
       interestRate = interestRate + .25;
       System.out.printf( "I borrowed %d at %.2f interest.\n", loanAmount,
```

```
                                                        interestRate );
    }//end public static void main(String arg[])

}//end public class variables
```

As we can see *interestRate* is a global variable that can accessed and changed in every method.

In all methods we have a method called loanAmount, and even thought they have the same name the amounts are different therefore they are different, and they can not see each other because they are at different address locations in RAM (see RAM Memory section).

Reading The Keyboard Using Scanner Class

We need to be able to communicate with the keyboard. The easiest way is to use the Scanner Class.

In the following program we will obtain input from the keyboard of three different data types: int, double and String.

```
/*
Author  : Michael Robinson
Program : scannerClass.java
Purpose : To present the Scanner class to read from KB
          Scanner myScanner = new Scanner(System.in);
                                                RETURNS
          byte x   = myScanner.nextByte();      byte
          double x = myScanner.nextDouble();    double
          int x    = myScanner.nextInt();       int
          float x  = myScanner.nextFloat();     float
          String x = myScanner.nextLine();      String
          long x   = myScanner.nextLong();      long
          short x  = myScanner.nextShort();     short

Updated : May 7, 2011
*/

import java.util.*;
import java.util.Scanner;

public class scannerClass
{
   public static void readTheKeyboard()
   {

      //create an object to read the keyboard
      //class   object                  method
      Scanner myScanner = new Scanner(System.in);

      System.out.print("Hi I am Scanner, please enter your name : ");
```

```
String name = myScanner.nextLine(); //user input get assigned to name

System.out.printf("Hi %s, nice meeting you\n", name);

System.out.print("\nNow we are going to do some math.");

double fnum, answer;
int snum;

System.out.print("\nEnter first number : ");

fnum =  myScanner.nextDouble(); //user input get assigned to fnum

System.out.print("Enter second number: ");

snum =  myScanner.nextInt();  //user input get assigned to snum

System.out.printf("\nAdding       %.2f + %d = %.2f\n",
                                  fnum, snum, (fnum + snum) );

answer = fnum * snum;

System.out.println("Multiplying: " + fnum + " by " + snum +" = " + answer );

String input;            //to hold a string line of input
char   singleCharacter;  //to hold a single character from String input

//ask user a question
System.out.print("\n\n\tAre we having fun? Y=Yes N=Not ? ");


//JAVA PROBLEMS:
//this line is needed to clear the keyboard buffer
//READS KB BUFFER WHICH HAS CR FROM PREVIOUS myScanner.nextInt()
//myScanner.nextInt() and myScanner.nextDouble DOES NOT read the
//CR at end of input, but reads the CR at beginning, left from last Scanner
input = myScanner.nextLine(); //clears CR at beginning of keyboard buffer
input = myScanner.nextLine(); //reads user keyboard input

//takes first character from user input and makes it uppercase
singleCharacter = input.toUpperCase().charAt(0);


while( singleCharacter != 'Y' && singleCharacter != 'N' )
{
   System.out.println("\tYou must be dreaming, just answer Yes or No.");

   //ask user a question
   System.out.print("\n\n\tAre we having fun? Y=Yes N=Not ? ");

   input = myScanner.nextLine(); //reads user keyboard input

   //takes first character from user input and makes it uppercase
   singleCharacter = input.toUpperCase().charAt(0);
}
```

```
      System.out.println("\n\tYour FULL answers is         : " + input);

      System.out.println("\tYour single Letter answer is : " + singleCharacter);

      if( singleCharacter == 'Y' )
      {
         System.out.println("\tI am very happy to hear that.");
      }
      else //if( singleCharacter == 'N' )
      {
         System.out.println("\tI am sorry to hear that.");
      }

   }//end public static void readTheKeyboard()


   public static void main(String arg[])
   {
      readTheKeyboard();
      System.out.println("\n\nEnd of program\n");

   }//end   public static void main(String arg[]){

}//end public class scannerClass {
```

This is the output:

```
Hi I am Scanner, please enter your name : Mike
Hi Mike, nice meeting you

Now we are going to do some math.
Enter first number : 45
Enter second number: 2

Adding       45.00 + 2 = 47.00
Multiplying: 45.0 by 2 = 90.0


      Are we having fun? Y=Yes N=Not ? y

      Your FULL answers is         : y
      Your single Letter answer is : Y
      I am very happy to hear that.


End of program
```

Since this program is very well documented, please read the program carefully to understand what each line does.

//end **Chapter Four : Building a house**

Chapter Five : if commands

If command

In my opinion the most used command in programming is the "if" command. Programming is the home of mind that is always making decisions, always asking what if?

The great thing about the implementation of the "if" command is that it is almost identical in every programming language.

Plain if

A plain if is very useful and very easy to implement.
The if statement has the following structure:

```
if( condition )
{
   commands
}
```

If the condition is TRUE then the commands between the curly brackets will be executed e.g.

```
float temperature = 92.7; //create and load a float variable

if( temperature > 90 )
{
   System.out.println( " Oh man it is hot!!" );
}
```

The output is:
Oh man it is hot!!

Let's examine this code:

```
public static void if_if( int grade )
{
   if( grade  >= 95 )
   {
      System.out.println("Your current grade is A == " + grade);
   }
   if( grade  >= 90 )
   {
      System.out.println("Your current grade is A- == "+ grade);
```

```
        }
        if( grade  >= 87 )
        {
           System.out.println("Your current grade is B+ == "+ grade);
        }
        if( grade  >= 83 )
        {
           System.out.println("Your current grade is B == "+ grade);
        }
        if( grade  >= 80 )
        {
           System.out.println("Your current grade is B- == "+ grade);
        }
        if( grade  >= 76 )
        {
           System.out.println("Your current grade is C+ == "+ grade);
        }
        if( grade  >= 70 )
        {
           System.out.println("Your current grade is C == "+ grade);
        }
        if( grade  >= 66 )
        {
           System.out.println("Your current grade is C- == "+ grade);
        }
        if( grade  >= 60 )
        {
           System.out.println("Your current grade is D+ == "+ grade);
        }
        if( grade  >= 56 )
        {
           System.out.println("Your current grade is D == "+ grade);
        }
        if( grade  >= 51 )
        {
           System.out.println("Your current grade is D- == "+ grade);
        }
        if( grade  <= 50 )
        {
           System.out.println("Your current grade is F == "+ grade);
        }
     }//end if_if()
```

If the grade passed to this method was 61, the output is:

```
Your current grade is D+ == 61
```

```
Your current grade is D == 61
Your current grade is D- == 61
```

because the grade 61 meets all the following conditions:

```
if( grade  >= 60 )
if( grade  >= 56 )
if( grade  >= 51 )
```

if_else

To solve this problem “if” has a sidekick called else, they work very well together, their structure is:

```
if( condition )
{
   commands
}
else
{
   commands
}
```

If the “if” condition is NOT true then else takes charge and its commands are executed, e.g.

```
float temperature = 42.7; //create and load a float variable

if( temperature > 80 )
{
   System.out.println( “ Oh man it is hot!!” );
}
else
{
   System.out.println( “ Summer time is gone!!” );
}
```

The output is:

```
 Summer time is gone!!
```

Now it gets better, there is another sidekick called “elseif”

if elseif else

There are times that the if and else combination is not enough, then we use the “if elseif else” combination e.g.

```
float temperature = 31.9; //create and load a float variable

if( temperature > 80 )
{
   System.out.println( " Oh man it is hot!!" );
}
elseif( temperature > 32 )
{
   System.out.println( " Summer time is gone!!" );
}
else
{
   System.out.println( " Oh man it is freezing!!" );
}
```

The output is:
```
 Oh man it is freezing!!
```

The key here is that every elseif and else belongs to the closest previous if command. Lets examine the following code:

```
public static void if_elseIf_elseBetter( int grade )
{
   if( grade  >= 95 )
   {
       System.out.println("Your current grade is A == " + grade);
   }
   else if( grade  >= 90 )
   {
       System.out.println("Your current grade is A- == "+ grade);
   }
   else if( grade  >= 87 )
   {
       System.out.println("Your current grade is B+ == "+ grade);
   }
   else if( grade  >= 83 )
   {
       System.out.println("Your current grade is B == "+ grade);
   }
   else if( grade  >= 80 )
   {
       System.out.println("Your current grade is B- == "+ grade);
   }
   else if( grade  >= 76 )
   {
       System.out.println("Your current grade is C+ == "+ grade);
   }
   else if( grade  >= 70 )
   {
       System.out.println("Your current grade is C == "+ grade);
```

```
        }
        else if( grade  >= 66 )
        {
            System.out.println("Your current grade is C- == "+ grade);
        }
        else if( grade  >= 60 )
        {
            System.out.println("Your current grade is D+ == "+ grade);
        }
        else if( grade  >= 56 )
        {
            System.out.println("Your current grade is D == "+ grade);
        }
        else if( grade  >= 51 )
        {
            System.out.println("Your current grade is D- == "+ grade);
        }
        else // grade  <= 50
        {
            System.out.println("Your current grade is F == "+ grade);
        }
    }//end if_elseIf_elseBetter()
```

If the grade passed to this method was 89, the output is:

```
Your current grade is B+ == 89
```

When the program enters the section of code:

```
        else if( grade  >= 87 )
```

it executes the code inside its curly brackets { } and skips all other elseif, else statements terminating the entire block of if elseif statements.

nested if

The nested if is very powerful and sometime complex, this is an example:

```
   public static void ifNested()
   {
      int number = 20;

      if( number > -1 )
      {
         if( number > 1 )
         {
             number++;
```

```
            if( number < 1 )
            {
               System.out.println( number );
            }
            else
            {
               System.out.println( "number is > 1 : " +  number );
            }
         }
         else
         {
            System.out.println( "the number is < 1 : " + number );
         }
      }
      else
      {
         System.out.println( "the value of number is : " + number );
      }

   }//end public static void ifNested()
```

Let's examine this code

```
      //assign the value 20 to the variable number
      int number = 20;
      //since number > -1 we follow this if
      if( number > -1 )
      {
         //here number is still > 1, so follow this if
         if(number > 1)
         {
             //add one to variable number, now it is 21
             number++;

             //this if fails because number is > 1
             if( number < 1)
             {
                System.out.println( number );
             }
             //so we follow this else,
             //since this else is true we terminate the entire
             //logic at this location and print this result
             else
             {
                System.out.println( "number is > 1 : " +  number );
```

```
            }
         }
         else
         {
            System.out.println( "the number is < 1 : " + number );
         }
      }
      else
      {
         System.out.println( "the value of number is : " + number );
      }

   }//end public static void ifNested()
```

This is the output:

```
   number is > 1 : 21
```

ternaryIf

The ternary(three way) "if" replaces some implementations of if elseif statements

```
         expression1 ? expression2 : expression3

         if   expression1 is true
         then expression2 is selected
         else expression3 is selected
```

other way to look at it is:

```
           condition ? value_if_true : value_if_false
```

```
  public static void trinaryIf()
  {
     int grade = 96;
     System.out.print("Grade: "+(grade < 70 ? "Fail." : "Pass."));
  }//end public static void trinaryIf()
```

Let's break it down

```
               (grade < 70 ? "Fail." : "Pass.")

             if grade < 70
                              Fail is printed
```

```
            else
                                Pass is printed
```

switch

I look at the switch statement as a variation of the if statement.

```
/*
Author  : Michael Robinson
Program : swithClass.java
Purpose : To present the switch statement

Updated : Jan 29, 2011
*/

import java.util.Scanner;

public class switchClass
{
   public static void switchOne()
   {
      //using numbers
      System.out.print("\nEnter your grade (60,70,80,90 or 0 to exit) : ");
      int grade = 999;

      Scanner keyboard = new Scanner(System.in); //request input from user

      grade = (int)keyboard.nextInt(); //convert input to int data type

      while(grade != 0)
      {
         switch(grade) //execute switch using the grade entered by the user
         {
            case 90:      //if grade = 90
                  System.out.println("You obtained " + grade);
                  break; //terminate loop
            case 80:      //if grade = 80
                  System.out.println("You obtained " + grade);
                  break; //terminate loop
            case 70:      //if grade = 70
                  System.out.println("You obtained " + grade);
                  break; //terminate loop
            case 60:      //if grade = 60
                  System.out.println("You obtained " + grade);
                  break; //terminate loop
            default:     //else
                  System.out.println("Enter 60, 70, 80, 90 or 0 only");
         }//end switch

         System.out.print("\nEnter your grade (60,70,80,90 or 0 to exit) : ");
         grade = (int)keyboard.nextInt(); //convert input to int data type

      }//end while(grade !=0)
```

```
   }//end public static void switchOne()
```

This is the output:

```
Enter your grade (60,70,80,90 or 0 to exit) : 99
Enter 60, 70, 80, 90 or 0 only

Enter your grade (60,70,80,90 or 0 to exit) : 0
```

```
   public static void switchTwo()
   {
      //using chars, leaving break out until the last case to show what happens
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter your program1 grade: ");
      String s = sc.next();   //request input from user
      char   p = s.charAt(0); //convert input to char data type

      String details = "";

      switch(p)
      {
         case 'F':       //if p == 'F'
         case 'f':       //if p == 'f'
                 details += "F";
         case 'D':       //if p == 'D'
         case 'd':       //if p == 'd'
                 details += "D";
         case 'C':       //if p == 'C'
         case 'c':       //if p == 'c'
                 details += "C";
         case 'B':       //if p == 'B'
         case 'b':       //if p == 'b'
                 details += "B";
         case 'A':       //if p == 'A'
         case 'a':       //if p == 'a'
                 details += "A";
                 break;  //terminate loop
         default:
                 details = s;
                 break;  //terminate loop
      }

      System.out.println("\nNice work on getting " + details +
                         " on your First program");

   }//end public static void switchTwo()
```

```
Enter your program1 grade: f
```

```
Nice work on getting FDCBA on your First program

      End of Program
```

The reason for getting FDCBA as final grade is because we do not have a “break” on each case as it is required.

```
   public static void main(String arg[])
   {
      switchOne();
      switchTwo();
      System.out.println("\n\tEnd of Program");

   }//end public static void main(String arg[])

}//end public class switchClass {
```

Limitations of the switch command

The limitations of the switch class is that it only accepts as input, integers or characters, while the “if” command accepts any type and amount of conditions.

//end Chapter Five : if commands

Chapter Six : Loops

Loops

In Java we have three types of loops, for, while, and do while loops.

for loops

There are two types of for loops in Java, the standard for loop which has three parts and the enhanced for loop (see Enhanced for Loops at the end of the arrays section).

The standard for loop looks as follows:

```
for( start ; condition ; step )
{
    commands
}
```

The for loop requires a counter variable, in most cases the int x variable is selected.

The **start** section determines the beginning of the loop, the loop can start at any location, 0, the end value (e.g. 5) or any value in between.

```
int x =0;
for(x = 0;
for(x = lastValue;
for(x = lastValue/2;
```

The **condition** section tells the loop that when such condition is met to terminate the loop. e.g.:

```
for(x = 0; x <= 5
```

The **step** section tells the for loop that every time it does a loop, execute whatever command is in the step. e.g.:

```
for(x = 0; x <= 5; x++)
```

x++ means add one to the current value of x.

Notice that we must place a ; after the first and second conditions.

Now that we have the heading or control line of the “for loop”, we add its body:

```
for(x = 0; x <= 5; x++)
{
   System.out.print( x + “ “ );
}
```

This is the output:
0 1 2 3 4 5

Notice that the program is checking if x < 0 and then if x = 5, so it is doing two instructions every time it does a loop, we can reduce these instructions to half by modifying the for loop header as follows:

```
for(x = 0; x < 6; x++)
{
   System.out.print( x + “ “ );
}
```

so now it only checks if x < 6, which gives us the same result as checking for <= 5.

To do this same process backwards we change the header as follows:

```
for(x = 5; x > -1; x--)
{
   System.out.print( x + “ “ );
}
```

This is the output:
5 4 3 2 1 0

To the **step** we can applied any operator we want:

x+=3, x-=1, x*=2, x/=2, x%=2, etc.

Of the three loops in Java, I think that the “for loop” is the most powerful and easier to control.

while loops

The while loop looks like this:

```
while( condition )
```

```
{
    commands
}
```

So if we compare the while loop with the for loop, what goes between the () in the while is the condition section of the for loop, but it is much more flexible and therefore powerful.

Assume we have variables that control the temperature, humidity, and altitude and we want to do a task while these three conditions meet some values.

```
int temperature = 50;
int humidity    = 10;
int altitude    = 0;

while( ((temperature > 0) && (humidity < 100) && (altitude>0)) ||
       ((temperature < 100) && (humidity > 0) && (altitude>0)) )
{
   //fly away
}
```

We can have as many conditions inside the () as needed.

Boolean conditions in while loops

When we studied the primitive data type we learned about the boolean data type. The while loops use the boolean values, either true or false.
This while loop uses a boolean type in the condition.

```
int temperature = 0;
boolean x = false;

System.out.print( "Temperature is normal" );

while( x != true)
{
   System.out.print( "." );
   if( temperature  == 50 )
   {
   System.out.println( "\nYour computer's temperature is above 99 degrees" );
   System.out.println( "Turning computer off .....\n" );
   x = true; //will reset condition to true and terminate loop
}
temperature++;
```

This is the output:
Temperature is
normal.....

```
Your computer's temperature is above 99 degrees
Turning computer off .....
```

Endless while loop

```
//this is an endless while loop
int counter = 0;

while( true )
{
   if( counter > Math.pow(10.0, 2.0))
   {
      System.out.println( "\nCounter = " + counter );
      break; //terminate while loop
   }
   counter++;
}
```

The break command terminates any type of loop.

This is the output:

```
Counter = 101
```

do while loops

The do while loop looks like this:

```
do
{
   //fly away
}
while( ((temperature > 0) && (humidity < 100) && (altitude>0)) ||
       ((temperature < 100) && (humidity > 0) && (altitude>0)) )
```

The difference between the while and the do while is that in the do while the process between the {} gets done at least once, but in the regular while() it gets done only if the condition is met, in other words “if the condition is true”.

Just like in the regular while loop, we can use boolean and endless loop conditions in the “do while” loops.

Enhanced for loops (summary)

These loops will be described at the end of next chapter. We need to know Arrays to understand this type of loops.

//end Chapter Six : Loops

Chapter Seven : Data Structures - Arrays

Data Structures

As we know data comes in many forms, int, float, Strings, etc. Data needs to be placed inside specific structures, such places are called Data Structures, once in there we can use the data as needed. One of the most used Data Structures in Computer Science is the Array.

Arrays

Arrays are data structures that group data items of the same data type. Arrays can contain data of one data type at the time such as int, char, float, double or any other Primitive data type, but we can not mix data types in the same array. We can also declare arrays of Wrapper data types such as Integer, Character, Float, Double, etc., or String data type. Arrays of Objects is a special case which will be addressed at the end of this chapter.

Arrays can be of one or multiple dimensions. They are very useful, but they have two short comings, first arrays can not have mixed data types, and once declared, their size can not be changed, we can not add or delete indexes, however the data inside each index can be changed.

One dimension array

We refer to a location in an array as "the index" and we always name the row first then the column. If the data is in row 0 and column 3, we say that the data is at index location 0,3. "Red Cross" will help us rebember R,C for Row,Column.

The following array has only one row which is row 0, and five columns as follow:

	col 0	col 1	col 2	col 3	col 4
row 0	132	115	165	103	176

Index	0,0	0,1	0,2	0,3	0,4
Contains	132	115	165	103	176

This could be a row of stores on a one story tall strip mall.

We can also look at this array as follows:

row , col	contains
0 0	132
0 1	115
0 2	165
0 3	103
0 4	176

Declaring/Creating one dimension arrays

Now lets create one dimensional arrays:

```
int    arrayOne[];    //creates the array arrayOne to hold ints
float  arrayTwo[];    //creates the array arrayTwo to hold floats
double arrayThree[];  //creates the array arrayThree to hold doubles
String arrayFour[];   //creates the array arrayFour to hold Strings
```

Allocating space to declared arrays

When we declare an array as above, we create it, but we do not allocate space for it in RAM memory. The size of an array is specified when we allocate space for it, as follows:

```
arrayOne[]   = new int[3];    //creates space for 3 elements
arrayTwo[]   = new float[2];  //creates space for 2 elements
arrayThree[] = new double[4]; //creates space for 4 elements
arrayFour[]  = new String[2]; //creates space for 2 elements
```

Declare and allocate at the same time

We can also declare(create) an array and assign it space at the same time as follows:

```
int    arrayOne[]   = new int[3];
float  arrayTwo[]   = new float[2];
double arrayThree[] = new double[4];
String arrayFour[]  = new String[2];
```

When we have an array with 4 indexes

```
double arrayThree[] = new double[4];
```

we say that its length is 4, with indexes 0, 1, 2, and 3, therefore the indexes go from 0 to length-1.

One of the most common mistakes when accessing an array is trying to access an index at location “length”, this will give us the error “index out of range”, we must rebember that indexes begin at location 0 and end at location “length – 1”, always!

Create, allocate and load data into arrays

```
    int    arrayOne[]    = {43,235,101};
or  int    arrayOne[3]   = {43,235,101};
```

Creates a single dimension array named arrayOne of int data type placing the values 43 at index 0, 235 at index 1,and 101 at index 2, again the length of arrayOne is 3 with indexes 0, 1, and 2.

If you declare the array as arrayOne[] and assign values to it, it knows how many indexes to create. You can also create the array giving it a size : arrayOne[3]

The previous rules apply to the following arrays:

```
    float  arrayTwo[]    = {12.54,4.23};
or  float  arrayTwo[2]   = {12.54,4.23};

    double arrayThree[]  = {2.33,2.31,9.0,7.12};
or  double arrayThree[4] = {2.33,2.31,9.0,7.12};

    String arrayFour[]   =  {“Joe”, “Smith”};
or  String arrayFour[2]  =  {“Joe”, “Smith”};
```

Load data to a declared array

We can also load data directly into each index as follows:

```
    arrayOne[0] =  43;
    arrayOne[1] = 235;
    arrayOne[3] = 101;

    arrayTwo[0] = 12.54;
    arrayTwo[1] =  4.23;

    arrayThree[0] = 2.33;
    arrayThree[1] = 2.31;
    arrayThree[2] = 9.0;
    arrayThree[3] = 7.12;

    arrayFour[0]  =  { “Joe” };
```

```
arrayFour[1]  =  { “Smith” };
```

Note: String arrays are themselves arrays of characters
arrayFour[0] is contains an array of 3 characters = Joe

If we wanted to load an array of length 1,000,000 with its corresponding location values e.g. array[55] = 55; We will need 1,000,000 lines of code, if we follow the above manual samples, but thanks to the for loop we only need 3 lines of code.

```
int x;
for( x = 0; x < 1,000,000; x++)
{
   array[x] = x;
}
```

Print all values inside this array

```
for( x = 0; x < 1,000,000; x++)
{
   System.out.printf( “%d\n”, array[x]);
}
```

Multi Dimensional arrays

Arrays of two or more dimensions are called Multidimensional Arrays.

Two Dimension Arrays

A two dimensional array looks as follows:

	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	**0,0**	0,1	0,2	0,3	**0,4**
Row 1	1,0	**1,1**	1,2	**1,3**	1,4
Row 2	2,0	2,1	**2,2**	2,3	2,4
Row 3	3,0	**3,1**	3,2	**3,3**	3,4
Row 4	**4,0**	4,1	4,2	4,3	**4,4**

In this graph we have a 5 x 5 array, it has 5 rows and 5 columns, from 0 to 4 each. Each location is called the index of the array.

In Computer Science we always start counting at 0. Here indexes 0 to

4 contain 5 indexes, therefore the size of this array is 5x5.

Every location is referred by the row,col location, Red Cross, RC cola, and RiCola are words that I have used in class to help remember that the ROW always comes before the COLUMN.

The r,c diagonal locations in a 5x5 array from 0,0 to 4,4 are:

r,c 0,0	r,c 1,1	r,c 2,2	r,c 3,3	r,c 4,4
0,0				
	1,1			
		2,2		
			3,3	
				4,4

The r,c diagonal locations in a 5x5 array from 0,4 to 4,0 are:

r,c 0,4	r,c 3,1	r,c 2,2	r,c 1,3	r,c 4,0
				4,0
			1,3	
		2,2		
	3,1			
0,4				

When dealing with data we always need to analyze the meaning of such data, in this example if we analyze the r,c of diagonal indexes from left,top to bottom,right locations we see that the rows == cols.

The indexes of the other diagonal indexes (top,right to bottom left) when we add the row+col of each index the totals are always 4.

Why is this important? In computers Science there is a discipline called Algorithms, the difference of a working program and a very efficient program is that the efficient program is done with efficient algorithms, so if we learn to analyze the data we are using, it will help us to come up with very efficient algorithms.

The following code will declare a two dimensional **int** array named

numbers2d as well as two variables, row and col.

Using a **for** loop we will load into each array location the addition of its row + col location.

The last section displays the contents of the entire array.

Two dimensional int arrays

```
//declares a 10x10 two dimensional array
int numbers2d[][] = new int[10][10];

int is the type of data in the array
numbers2d is the array's name
[][] describes that it is a two dimensional array
new int[10][10] means a new array with 10 rows and 10 columns

//Declare and initialize local variables
int row = 10, col = 10;
int x = 0, y = 0;

//Load two dimensional array with each location's row+col
System.out.println( "\nLoading two dimensional array" );
for(x=0; x<row; x++)
{
   for(y=0; y<col; y++)
   {
      numbers2d[x][y] = x+y;
   }
}


//Display the contents of the two dimensional array
System.out.println("  \nTwo dimensional array' contents");
for(x=0; x<row; x++)
{
   for(y=0; y<col; y++)
   {
      if( (x+y) < col )
      {
         System.out.print( " " );
      }
      System.out.print(" “+System.out.print( numbers2d[x][y] );
   }
   System.out.println();
}
```

This is the output:

```
                    Loading two dimensional array

                  Two dimensional array' contents
                   0  1  2  3  4  5  6  7  8  9
                   1  2  3  4  5  6  7  8  9 10
                   2  3  4  5  6  7  8  9 10 11
                   3  4  5  6  7  8  9 10 11 12
                   4  5  6  7  8  9 10 11 12 13
                   5  6  7  8  9 10 11 12 13 14
                   6  7  8  9 10 11 12 13 14 15
                   7  8  9 10 11 12 13 14 15 16
                   8  9 10 11 12 13 14 15 16 17
                   9 10 11 12 13 14 15 16 17 18

The diagonal values above from locations 0,0 to 9,9 are:
                    0 2 4 6 8 10 12 14 16 18 = 90


The diagonal values above from locations 9,9 to 0,0 are:
                        9 9 9 9 9 9 9 9 9 = 90
```

Multidimensional arrays do not need to be perfect squares such as this one that is 10x10, they can have different amount of rows and columns.

To manually load data into a multidimensional array is a little tricky:

```
   //declares an initializes a 10x10 two dimensional array
   int numbers2d[][] = {  //each {} section represents a row
                       {0,  1,  2,  3,  4,  5,  6,  7,  8,  9},
                       {1,  2,  3,  4,  5,  6,  7,  8,  9, 10},
                       {2,  3,  4,  5,  6,  7,  8,  9, 10, 11},
                       {3,  4,  5,  6,  7,  8,  9, 10, 11, 12},
                       {4,  5,  6,  7,  8,  9, 10, 11, 12, 13},
                       {5,  6,  7,  8,  9, 10, 11, 12, 13, 14},
                       {6,  7,  8,  9, 10, 11, 12, 13, 14, 15},
                       {7,  8,  9, 10, 11, 12, 13, 14, 15, 16},
                       {8,  9, 10, 11, 12, 13, 14, 15, 16, 17},
                       {9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
                       };
```

Arrays have many methods, here we will use the array.length method

```
//Display a two dimensional array using array's length method
System.out.println( "  \nTwo dimensional array's contents" );
System.out.println( "    using the array's length " );
for(x=0; x < numbers2d.length; x++)
{
   for(y=0; y<col; y++)
   {
      if( (x+y) < 10)
      {
         System.out.print( " " );
      }
      System.out.print( " " + numbers2d[x][y] );
   }
   System.out.println();
}
```

Two dimensional String arrays

```
public static void stringLoadDisplay2d()
{
   String names[][] = {  { "Joe", "Smith" },
                         { "Maria", "Perez" },
                         { "James", "Einstein" }
                      };

   int x = 0;
   int y = 0;
   for(x=0; x < names.length; x++)
   {
      for(y=0; y<2; y++)
      {
         System.out.print( " " + names[x][y] );
      }
      System.out.println();
   }
}
```

The output will be:

```
Joe Smith
Maria Perez
James Einstein
```

Three Multidimensional Arrays

This is the representation of an apartment building:

```
   int floors   = 4; //floors
   int rows     = 5; //rows
   int columns  = 3; //columns

   int array3D[][][] = new int[floors][rows][columns];
   int x,y,z;

   System.out.println(" This is a building with " + floors + " Floors" );
   System.out.println(" Using a 3 Dimensional Array\n" );

   for(x=0; x<floors; x++)
   {
      System.out.println(" Floor = " + (x+1));
      for(y=0; y<rows; y++)
      {
         for (z=0; z<columns; z++)
         {
            array3D[x][y][z]= (x+10)+(y*10)+z;
            if( (x + y + z) < 10 )
            {
               System.out.print(" ");
            }
            System.out.print( (x+1) + "" + array3D[x][y][z] + " " );
         }
         System.out.println();
      }
      System.out.println();
   }
   System.out.println();
```

The output is:

```
 This is a building with 4 Floors
 Using a 3 Dimensional Array

 Floor = 1
 110  111  112
 120  121  122
 130  131  132
 140  141  142
 150  151  152

 Floor = 2
 211  212  213
 221  222  223
 231  232  233
 241  242  243
 251  252  253
```

```
Floor = 3
312  313  314
322  323  324
332  333  334
342  343  344
352  353  354

Floor = 4
413  414  415
423  424  425
433  434  435
443  444  445
453  454  455
```

Now that we know what arrays are, lets look into the main method:

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

As we know the main method in Java is the starting method for all Java programs. So you probably have been asking yourself what is the purpose of the “String arg[]” parameter that is being passed to the main method?

As you progress in you programming career you will find that there are times that because some languages are better than others for a particular job, sometimes we need to use multiple languages to accomplish a task.

Java uses the parameters in the main method to receive data from other programs even if they are written in a different language. String arg[] is just a single dimension array of String data type which allows us to pass as many string variables as we want to the Java main method.

Parallel Arrays

Parallel Arrays are used to combine multiple arrays that might contain different data types.

```
int    studentID[]      = { 50102, 23908, 12098 };
String studentName[]    = { “Joe Smith”, “Ann Pope”, “Mike Rome” };
float  studentBalance[] = { 10.00, 123.78, 1.07 };
```

Here we have three arrays, each containing three items, and each array holds different type of data, int, String and float.

We need to create a report showing the student's name, id and current balance owed. To create this report we use what is called Parallel Arrays. The only requirement is that all arrays most have the same amount of rows, in this case there are three rows in each array.

This is one of several way to write Parallel arrays.

```
for(int x=0; x<studentID.length; x++)
{
   System.out.print( studentName[x] + "\t" +
                     studentID[x] + "\t" +
                     studentBalance[x]
                   );
}
```

This will produce the following output:

```
Joe Smith     50102     10.00
Ann Pope      23908     123.78
Mike Rome     12098     1.07
```

You can use as many arrays as needed to produce a Parallel Arrays.

Enhanced for Loops

Now that we know how Arrays work, lets see what Enhanced for Loops are.

Enhanced for Loops are a variation of the “for loop” that is used with Arrays and ArrayLists which we will see next, this is an example:

Using the following String Array:

```
String array1[] = {"one", "square", "two ","three", "seven" };
```

The enhanced for loop looks like this:

```
for( String elements : array1 )
{
   System.out.printf(" %s", elements);
}
```

Inside the parenthesis of the for loop, we create a String variable, in this case I call it “elements”, then we place the : symbol, after that the name of the Array that we want to access, in this case

array1.

This line of code Iterates through the array sequentially from the beginning to the end. We do not need to know the length of the array. The value of each index in the array is assigned to the variable “element”.

//end **Chapter Seven : Data Structures - Arrays**

Chapter Eight : Data Structures – ArrayList

ArrayList

Arrays have a size limitation, once an array declares its size it can not be modified. We can not add or delete rows or columns to/from it.

ArrayLists are the solution. ArrayLists are another type of Data Structure; we can modify the ArrayList's size at any time, we can add or delete indexes from any location, ArrayLists are dynamic.

Let's create an ArrayList containing String Type data as follows:

```
ArrayList<String> arrayList = new ArrayList<String>();
```

Now we create an ArrayList containing Integer Type data as follows:

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
```

Notice the type of data in the second ArrayList is <Integer> instead of <int>, the reason is that ArrayLists require data type with methods and the Primitive Data types DO NOT have methods so we need to use the wrappers data types which DO have methods.
This is an ArrayList using for loops:

```
/**
Author  : Michael Robinson
Program : arrayListSimpleClass.java
Purpose : How to create a Java ArrayList using for loops.
          Shows how to add elements to an ArrayList
          How to delete items
          How to get and print data items using for loops.

Updated : June 29th, 2012

*/

import java.util.ArrayList;

public class arrayListSimpleClass
{
   public static void alphaArrayList()
   {
      System.out.println("-----------------------------------------");
      System.out.println(" Processing ArrayList of String Data Type");
      System.out.println("              Using for Loops");
      System.out.println("-----------------------------------------");
```

```
   System.out.println(" Creating ArrayList...");
   //create an ArrayList object
   ArrayList<String> arrayList = new ArrayList<String>();

   System.out.println(" Adding data elements to the ArrayList...");
   arrayList.add("a");
   arrayList.add("b");
   arrayList.add("c");
   arrayList.add("d");
   arrayList.add("e");
   arrayList.add("f");

   System.out.println("\n Getting data elements from the ArrayList");

   //Use get method of ArrayList class to retrieve an element
   //get( index ) returns element at the index in the ArrayList

   for(int x=0; x<arrayList.size(); x++)
   {
      System.out.print(" "+arrayList.get(x));
   }

   //now remove elements !!! always remove from the highest to the lowest
   //location
   System.out.println("\n\n Removing data elements from the ArrayList on “ +
                      “index 3 = " + arrayList.get(3) +
                      "\n                                          and “ +
                      “index 1 = " + arrayList.get(1) );

   arrayList.remove(3);
   arrayList.remove(1);

   System.out.println("\n");
   System.out.println(" Getting data elements from ArrayList after removing");

   for(int x=0; x<arrayList.size(); x++)
   {
      System.out.print(" "+arrayList.get(x));
   }

   System.out.println("\n");

}//end public static void alphaArrayList()


public static void numericArrayList()
{
   System.out.println("-----------------------------------------");
   System.out.println(" Processing ArrayList of Integer Data Type");
   System.out.println("              Using for Loops");
   System.out.println("-----------------------------------------");

   System.out.println(" Creating ArrayList...");
   //create an ArrayList object
   ArrayList<Integer> arrayList = new ArrayList<Integer>();
```

```
      System.out.println(" Adding data elements to the ArrayList...");
      arrayList.add(1100);
      arrayList.add(2100);
      arrayList.add(3100);
      arrayList.add(4100);
      arrayList.add(5100);
      arrayList.add(6100);

      System.out.println("\n Getting data elements in the ArrayList");

      //Use get method of ArrayList class to retrieve an element
      //get( index ) returns element at the index in the ArrayList
      for(int x=0; x<arrayList.size(); x++)
      {
         System.out.print(" "+arrayList.get(x));
      }

      System.out.println("\n\n Removing data elements from the ArrayList on “ +
                         "index 3 = " + arrayList.get(3) +
                        "\n                                          and “ +
                         "index 1 = " + arrayList.get(1) );

      //now remove some !!! always remove from the height to the lowest location
      arrayList.remove(3);
      arrayList.remove(1);

      System.out.println("\n Getting data elements from ArrayList after remove");

      for(int x=0; x<arrayList.size(); x++)
      {
         System.out.print(" "+arrayList.get(x));
      }

   }//end public static void numericArrayList()


   public static void main(String args[])
   {
      alphaArrayList();
      numericArrayList();

   }//end public static void main(String args[])


}//end public class arrayListSimpleClass
```

This is the output:

```
----------------------------------------
 Processing ArrayList of String Data Type
             Using for Loops
----------------------------------------
 Creating ArrayList...
```

```
 Adding data elements to the ArrayList...

 Getting data elements from the ArrayList
 a b c d e f

 Removing data elements from the ArrayList on index 3 = d
                                          and index 1 = b


 Getting data elements from the ArrayList after removing
 a c e f

-----------------------------------------
 Processing ArrayList of Integer Data Type
             Using for Loops
-----------------------------------------
 Creating ArrayList...
 Adding data elements to the ArrayList...

 Getting data elements in the ArrayList
 1100 2100 3100 4100 5100 6100

 Removing data elements from the ArrayList on index 3 = 4100
                                          and index 1 = 2100

 Getting data elements from ArrayList after removing
 1100 3100 5100 6100
```

Iterator

The Iterator class is very powerful, in a way it is similar to the for loop. It allows us to see if we have data/elements in a data structure, move forwards element by element, and remove any element in the data structure. Iterator exists in several data structures, among them the ArrayList and the List.

When we use an iterator to go through a data structure, the items inside the structure are called elements.

This is how an Iterator is created and used.

First we need to create an instance of the Iterator, in this case we will call it itr:

```
//create an Iterator object using Iterator.
Iterator<Integer> itr = arrayList.iterator();
```

Here we created the “itr” iterator for the ArrayList called arrayList

that we created in the previous **ArrayList** section.

Java Iterators have three methods:

hasNext which returns a boolean primitive data type. If it returns true the iteration has found an element inside the data structure, if it returns false it means that it is at the end of the data structure and there are no more items in it e.g.:

```
//use hasNext() to iterate through the elements
System.out.println("Iterating through ArrayList elements...");
while(itr.hasNext())
{
   commands;
}
```

If the command "itr.hasNext()" returns true, we have determined that the data structure has data elements in it. Now we can extract those elements, one by one, and use them as follows:

```
//use hasNext() and next() methods of Iterator
//to iterate through the elements
System.out.println("Iterating through ArrayList elements...");
while(itr.hasNext())
{
   System.out.print(" "+itr.next());
}
```

Here the line "System.out.print(" "+itr.next());" will use the next() method of the iterator "itr", itr.next() to extract the next element in the data structure and print it.

Using Iterator in an ArrayList

The following program shows how to use Iterators in an ArrayList:

```
/**
Author  : Michael Robinson
Program : arrayList_Iterators.java
Purpose : How to create a Java ArrayList using Iterators.
          Shows how to add elements to an ArrayList
          How to delete items
          How to get and print data items using for loops.

Updated : July 2nd, 2012

*/
```

```
import java.util.ArrayList;
import java.util.Iterator;

public class arrayList_Iterators
{
   public static void alphaArrayList()
   {
      System.out.println("-----------------------------------------");
      System.out.println(" Processing ArrayList of String data type");
      System.out.println("           using ITERATORS               ");
      System.out.println("-----------------------------------------");

      System.out.println(" Creating ArrayList...");

      //create an ArrayList object
      ArrayList<String> arrayList = new ArrayList<String>();

      System.out.println(" Adding data elements to the ArrayList...");
      arrayList.add("a");
      arrayList.add("b");
      arrayList.add("c");
      arrayList.add("d");
      arrayList.add("e");
      arrayList.add("f");

      System.out.println("\n Getting data elements from the ArrayList");
      System.out.println(" Iterating through ArrayList elements...");

      //This is a loop type class which allows to loop thru the ArrayList
      Iterator<String> itr = arrayList.iterator();

      //use hasNext() and next() methods of Iterator to iterate through the
      //ArrayList
      while(itr.hasNext())
      {
         System.out.print(" "+itr.next());
      }

      //now remove elements !!! always remove from the highest to the lowest index
      System.out.println("\n\n Removing data elements from the ArrayList on “ +
                         “index 3 = " + arrayList.get(3) +
                         "\n                                        and “ +
                         “index 1 = " + arrayList.get(1) );

      arrayList.remove(3);
      arrayList.remove(1);

      System.out.println();
      System.out.println(" Getting data elements from ArrayList after remove");
      System.out.println(" Iterating through ArrayList elements...");

      itr = arrayList.iterator(); //reset iterator to beginning of ArrayList

      //use hasNext() and next() methods of Iterator to iterate through the
      //ArrayList
```

```
   while(itr.hasNext())
   {
      System.out.print(" "+itr.next());
   }

   System.out.println("\n\n");

}//end public static void alphaArrayList()


public static void numericArrayList()
{
   System.out.println("-----------------------------------------");
   System.out.println(" Processing ArrayList of Integer data type");
   System.out.println("           using ITERATORS               ");
   System.out.println("-----------------------------------------");

   System.out.println(" Creating ArrayList...");
   //create an ArrayList object
   ArrayList<Integer> arrayList = new ArrayList<Integer>();

   System.out.println(" Adding data elements to the ArrayList...");
   arrayList.add(1100);
   arrayList.add(2100);
   arrayList.add(3100);
   arrayList.add(4100);
   arrayList.add(5100);
   arrayList.add(6100);

   System.out.println("\n Getting data elements in the ArrayList");
   System.out.println(" Iterating through ArrayList elements...");

   //This is a loop type class which allows to loop thru the ArrayList
   Iterator<Integer> itr = arrayList.iterator();

   //use hasNext() and next() methods of Iterator to iterate through the
   //ArrayList
   while(itr.hasNext())
   {
      System.out.print(" "+itr.next());
   }

   System.out.println("\n\n Removing data elements from the ArrayList on " +
                      "index 3 = " + arrayList.get(3) +
                      "\n                                          and " +
                      "index 1 = " + arrayList.get(1) );

   //now remove some !!! always remove from the height to the lowest location
   arrayList.remove(3);
   arrayList.remove(1);

   System.out.println("\n Getting data elements from ArrayList after remove");
   System.out.println(" Iterating through ArrayList elements...");

   itr = arrayList.iterator(); //reset iterator to beginning of ArrayList
```

```
      //use hasNext() and next() methods of Iterator to iterate through the
      //ArrayList
      while(itr.hasNext())
      {
         System.out.print(" "+itr.next());
      }

   }//end public static void numericArrayList()


   public static void main(String[] args)
   {
      alphaArrayList();
      numericArrayList();
   }

}//end public class arrayList_Iterators
```

This is the output:

```
----------------------------------------
 Processing ArrayList of String data type
          using ITERATORS
----------------------------------------
 Creating ArrayList...
 Adding data elements to the ArrayList...

 Getting data elements from the ArrayList
 Iterating through ArrayList elements...
 a b c d e f

 Removing data elements from the ArrayList on index 3 = d
                                          and index 1 = b

 Getting data elements from ArrayList after remove
 Iterating through ArrayList elements...
 a c e f



----------------------------------------
 Processing ArrayList of Integer data type
          using ITERATORS
----------------------------------------
 Creating ArrayList...
 Adding data elements to the ArrayList...

 Getting data elements in the ArrayList
 Iterating through ArrayList elements...
 1100 2100 3100 4100 5100 6100

 Removing data elements from the ArrayList on index 3 = 4100
                                          and index 1 = 2100
```

```
 Getting data elements from ArrayList after remove
 Iterating through ArrayList elements...
 1100 3100 5100 6100
```

Now we want to test if there is data in an ArrayList.

```
    //Create iterator to see if the data structure is empty
    //use hasNext() and next() methods of Iterator to iterate through the elements
    Iterator<Integer> itr3 = arrayList.iterator();

    System.out.println(" Testing if there is data in an ArrayList");
    if( itr3.hasNext() )
    {
       while(itr3.hasNext())
       {
          System.out.print(" "+itr3.next());
       }
       System.out.println("\n");
    }
    else
    {
       System.out.println(" The data structure is empty\n");
    }
```

This is the output:

```
 Testing if there is data in the Data Structure
 The data structure is empty
```

Copying Arrays into an ArrayList Modify and Back

Remember that arrays have two shortcomings, an array contains only one type of data at the time, e.g. ints, floats, Strings, etc. Also when we declare an array we can not modify its size. Fortunately we can use the ArrayList to modify any array's size, by moving the array's data into an ArrayList, then we can modify the size of the ArrayList by adding or removing elements in it. Then once we get the required size we copy the ArrayList data back into one or multiple new Arrays.

The following program show us how to create multiple arrays, one ArrayList, Iterators, and to use the enhanced for loop.

```
import java.util.ArrayList;
import java.util.Iterator;

/**
Author  : Michael Robinson
Program : arrayToArrayListAndBack.java
```

```
Purpose : How to copy Array's data to an ArrayList
          Modify the ArrayList
          Copy ArrayList data to a new Array
Updated : December 10, 2011
*/

public class arrayToArrayListAndBack
{
   public static void arraysToListAndBack()
   {
      //create Array array1
      System.out.println("-Creating Array array1");
      String array1[] = {"one ", "square", "two ","three", "cube",
                         "seven"};

      //create an ArrayList object
      System.out.println("-Creating ArrayList list1");
      ArrayList<String> list1 = new ArrayList<String>();

      //add Array array1 to ArrayList list1
      System.out.println("\n-Copying Array array1 to " +
                         "ArrayList list");
      for( String elements : array1 )
      {
         list1.add( elements );
         System.out.printf(" %s", elements);
      }

      //create another Array array2
      System.out.println("-Creating Array array2");
      String array2[] = {"four", "triangle", "five", "cone", "six",
                         "eight"};

      //add Array array2 to ArrayList list1
      System.out.println("\n-Copying Array array2 to " +
                         "ArrayList list");
      for( String elements : array2 )
      {
         list1.add( elements );
         System.out.printf(" %s", elements);
      }
      System.out.println("\n");

      //create Iterator itr to travel through the Arraylist
      //list
      System.out.println("-Create Iterator itr to " +
```

```
                    “travel through the Arraylist list");
  Iterator<String> itr = list1.iterator();

  //use hasNext() and next() methods of Iterator to
  //iterate through the elements
  System.out.println("-Iterating through ArrayList “ +
                    “list1 elements...");
  while(itr.hasNext())
  {
     System.out.print(" "+itr.next());
  }
  System.out.println("\n");

  System.out.println("-Using the new Iterator itr2 we copy “ +
                    “the elements in the even locations “ +
                    "\n to a new array of length 10 called ” +
                    “array3, and the elements in the odd “ +
                    “locations to a new array of length 10 “+
                    “called array4" );

System.out.println("\n-Create Iterator itr2 to travel through “ +
                  “the Arraylist list." +
                  "\n In Java Iterators can NOT be reset “ +
                  “to the beginning of the list " +
                  "\n to travel the list again we must “ +
                  “create a new Iterator.");
  //create a new iterator itr2 to copy element to new bigger
  //arrays
  Iterator<String> itr2 = list1.iterator();

  System.out.println("\n-Creating two new arrays of length “ +
                    “10 each. array3 and array4");
  //create two arrays of size 10 each
  String array3[] = new String[10];
  String array4[] = new String[10];

  System.out.println("\n-Copying even elements in list1 to “ +
                    “array3 and odd elements to array4");
  //copy even elements in list1 to array3 and odd elements “ +
  //to array4
  int x=0;
  int y=0;
  int z=0;
  while(itr2.hasNext())
  {
     if( (x % 2)==0 ) //copy even elements in list to array3
```

```
   {
      array3[z] =  itr2.next();
      System.out.print(" " + " array3[" + z +"] = " + array3[z]
                       + "\t" );
      z++;
   }
   else  //copy odd elements in list to array4
   {
      array4[y] =  itr2.next();
      System.out.println( "array4["+ y +"] = " + array4[y]);
      y++;
   }
   x++;
}

z=0;
System.out.println("\n-Displaying the elements in list1" +
                   " of length "+ list1.size() );

for(x=0; x<list1.size(); x++)
{
   System.out.print( " " + list1.get(x) + "\t" );
   z++;
   if(z==2)
   {
      System.out.println();
       z=0;
    }
}

z=0;
System.out.println("\n-Displaying the elements in array3" +
                   " of length "+ array3.length);
for(x=0; x<array3.length; x++)
{
   System.out.print( " " + array3[x] + "\t" );
   z++;
   if(z==2)
   {
      System.out.println();
      z=0;
   }
}

z=0;
System.out.println("\n-Displaying the elements in array4");
```

```
      for(x=0; x<array4.length; x++)
      {
         System.out.print( " " + array4[x] + "\t" );
         z++;
         if(z==2)
         {
            System.out.println();
            z=0;
         }
      }
      System.out.println("\n");

   }//end arraysToListAndBack()


   public static void main(String arg[])
   {
      arraysToListAndBack();
   }

}//end public class arrayToArrayListAndBack
```

This is the entire output:

```
-Creating Array array1
-Creating ArrayList list1

-Copying Array array1 to ArrayList list
 one  square two  three cube seven-Creating Array array2

-Copying Array array2 to ArrayList list
 four triangle five cone six eight

-Create Iterator itr to travel through the Arraylist list
-Iterating through ArrayList list1 elements...
 one  square two  three cube seven four triangle five cone six eight

-Using the new Iterator itr2 we copy the elements in the even
 locations to a new array of length 10 called array3, and the
 elements in the odd locations to a new array of length 10 called
 array4

-Create Iterator itr2 to travel through the Arraylist list
 In Java Iterators can NOT be reset to the beginning of the list
 to travel the list again we must create a new Iterator.
```

```
-Creating two new arrays of length 10 each. array3 and array4

-Copying even elements in list1 to array3 and odd elements to array4
  array3[0] = one   array4[0] = square
  array3[1] = two   array4[1] = three
  array3[2] = cube  array4[2] = seven
  array3[3] = four  array4[3] = triangle
  array3[4] = five  array4[4] = cone
  array3[5] = six   array4[5] = eight

-Displaying the elements in list1 of length 12
 one       square
 two       three
 cube      seven
 four      triangle
 five      cone
 six  eight

-Displaying the elements in array3 of length 10
 one       two
 cube      four
 five      six
 null      null
 null      null

-Displaying the elements in array4
 square    three
 seven     triangle
 cone      eight
 null      null
 null      null
```

Notice that in array3 and array4 the last four indexes/locations contain the word null, this means that those locations are not initialize/used, they were created when we created them:

Object Data Types

We have been using Primitive, Wrappers and String data types from the beginning, in this chapter we used them to populate Arrays and ArrayLists. Now that we are experts in using these data types, we are going to learn about another data type called Objects (notice Objects with uppercase O).

Up to now, when we create an array or an ArrayList we make them to

accept only one type of data. This new data type called Object data type allows us to accept the mixture of all previous learned data types into an array or Arraylist.

The following program will teach us how to use Object Data Type.

```
/*
Author  : Michael Robinson
Program : ObjectsArraysArrayLists.java
Purpose : - To create an Array of Objects and
            copy it into an ArrayList of Objects
          - From a text file load Objects into an Array
            and display them

Updated : July 3nd, 2012
*/

import java.util.ArrayList;

public class ObjectsArraysArrayLists
{

   public static void arrayOfObjects()
   {
      System.out.println("\n Processing an array of Objects");

      //create an array of Objects
      Object arrayObjects[] =
          { 1, "one", 1.4, 0.25, "COP2250", "Java Language", 100 };

      //display elements in an array to Objects
      for(int x = 0; x < arrayObjects.length ; x++ )
      {
         System.out.println(" arrayObjects[" + x + "] = " +
                            arrayObjects[x]);
      }

      //call a method passing an array of Objects
      arrayListObjects( arrayObjects );

   }//end     public static void arrayOfObjects()


   public static void arrayListObjects( Object arrayObjects[] )
   {
```

```
      System.out.println("\n Copying an array of Objects into " +
                         "an ArrayList of Objects");

      //create an ArraList of Objects
      ArrayList<Object> arrayListObjects = new ArrayList<Object>();

      //copy the elements in an array of Objects to an ArraList of
      //Objects and display them
      int x = 0;
      for( x = 0; x < arrayObjects.length; x++ )
      {
         arrayListObjects.add(arrayObjects[x]);
         System.out.println(" arrayListObjects(" + x + ") = " +
                            arrayListObjects.get(x) );
      }

   }//end public static void arrayListObjects(Object arrayObjects[])


   public static void main(String arg[])
   {
      arrayOfObjects();

   }//end public static void main(String arg[])


}//end public class ObjectsArraysArrayLists
```

This is the output:

```
 Processing an array of Objects
 arrayObjects[0] = 1
 arrayObjects[1] = one
 arrayObjects[2] = 1.4
 arrayObjects[3] = 0.25
 arrayObjects[4] = COP2250
 arrayObjects[5] = Java Language
 arrayObjects[6] = 100

 Copying an array of Objects into an ArrayList of Objects
 arrayListObjects(0) = 1
 arrayListObjects(1) = one
 arrayListObjects(2) = 1.4
 arrayListObjects(3) = 0.25
 arrayListObjects(4) = COP2250
 arrayListObjects(5) = Java Language
 arrayListObjects(6) = 100
```

In the next chapter we are going to use objects to read and write data files.

//end **Chapter Eight Data Structures : ArrayList**

Chapter Nine : Data Structures – Files

Files

What is a file

Files are Data Structures that are used to store data.

For instance any program that we do in any language is saved to a disk, CD, etc under a document name, e.g. myFirstStringProgram.java This document is called a text file.

What is a record

Since we know what arrays are from Chapter Six : Data Structures – Arrays, we can say that a text file with one record is a single dimensional array, and that a text file with more than one record is a two dimensional array. In other words a record is a line of text in a data structure such as a file or an array.

Records such as:

```
Joe Smith CS
```

are composed of sections such as Joe, Smith, and CS. Each of these sections are called fields. A file can contain thousands of records, and a record can contain thousands of fields. It is limited by the type of data structure, the Operating System. With today's technologies we can have files that contain tera records, such as a benchmark data file that is used to test sorting algorithms.

Records are separated (delimited) in many different forms, for instance Spreadsheet use the tab symbol to delimit their fields, the above record example (Joe Smith CS) is delimited by the space symbol. There is a data structure called SQL databases, what we call files, SQL calls tables, records are named the same, and fields are called columns.

Catching Errors Exceptions In Data Files

When we create Java programs we create at least three different types of errors, the most common is called “Syntax error” which is a typographical error, e.g instead of writing print we write frint, next are “logical errors” in which we are telling the computer to do one thing when we think we are telling something else, the next one

are called "runtime errors" which only happen when we run the program, e.g: we try to open a file that does not exist.

In Java errors are called Exceptions, as we will see in the following sections every method that deals with files must have in their headings the command:

throws IOException

This command will take care of any type of error that has to do with files.

If we try to open a file that does not exist we will get an error like this:

```
At 4 - addAllValuesInFile() : Exception in thread "main"
java.io.FileNotFoundException: customer.txt (No such file or directory)
   at java.io.FileInputStream.open(Native Method)
   at java.io.FileInputStream.<init>(FileInputStream.java:137)
   at java.io.FileInputStream.<init>(FileInputStream.java:96)
   at java.io.FileReader.<init>(FileReader.java:58)
   at filesBasicClass.addAllValuesInFile(filesBasicClass.java:96)
   at filesBasicClass.main(filesBasicClass.java:152)
```

The problem here is that the program terminates/stops which is not always a desired result.

Java gives us ways to handle errors like this without terminating/stopping the program. We have a complete section that explains how to implement this process called Exceptions Handling in this chapter before section **Text Files using the Formatter Class.**

Text Files

Documents called text files, can be seen by any text editor or word processor

Java has several ways to create, write, read, and modify files. In the following program we use:

```
To Create and write to a new file;
FileWriter      customers = new FileWriter("customers.txt");
PrintWriter     output    = new PrintWriter( customers );

To Open an existing text file and append data to it;
FileWriter      customers = new FileWriter("customers.txt", true);
PrintWriter     output    = new PrintWriter( customers );
```

```
   To open and read and existing text file:
   FileReader      customers = new FileReader("customers.txt");
   BufferedReader  inputFile = new BufferedReader( customers );

Java uses other classes in additions to this ones, to create, append,
and read files, which we will see afterwards.


/**
Author  : Michael Robinson
Program : flesBasicClass.java
Purpose : Create new text file & write data into file
          Read and process data from text file
          Open existing text file, write and read data into/from file
          Also converts String to double and String to integer

   To Create and write to a new file;
   FileWriter      customers = new FileWriter("customers.txt");
   PrintWriter     output    = new PrintWriter( customers );

   To Open an existing text file and append data to it;
   FileWriter      customers = new FileWriter("customers.txt", true);
   PrintWriter     output    = new PrintWriter( customers );


   To open and read and existing text file:
   FileReader      customers = new FileReader("customers.txt");
   BufferedReader  inputFile = new BufferedReader( customers );

Updated : May 4, 2011

*/

import java.io.*;
import java.util.Scanner;

public class filesBasicClass
{
   //To Create and write to a new file;
   public static void openWriteNewFile() throws IOException
   {
      System.out.print("\n At 1 – openWriteNewFile()" +
                       " creating new file and adding: ");

      //note: customers.txt is the name of the file to be
      //written to the secondary storage (hard disk, CD, etc)
```

```
   //so
   FileWriter  customers = new FileWriter("customers.txt");
   PrintWriter output    = new PrintWriter( customers );


   for(int x = 10; x >=0; x--)
   {
       output.println(x);
       System.out.print( " " + x);
   }

   output.close(); //closes the file
   System.out.println("<---eof");

}//end public static void openWriteNewFile() throws IOException


public static void openWriteExistingFile() throws IOException
{
   System.out.print("\n At 2 - openWriteExistingFile() " +
                    "Appending data: ");
   FileWriter  customers = new FileWriter("customers.txt", true );
   PrintWriter output    = new PrintWriter( customers );

   for(int x = 10; x < 16; x++)
   {
       output.println(x);
       System.out.print( " " + x );
   }
   output.close();
   System.out.println("<---eof");

}//end public static void openWriteExistingFile()


//To open and read and existing text file:
public static void openReadExistingFile() throws IOException
{
   System.out.print("\n At 3 - openReadExistingFile() : ");
   FileReader      customers = new FileReader("customers.txt");
   BufferedReader  inputFile = new BufferedReader( customers );

   String inputFileRecord;

   inputFileRecord = inputFile.readLine();
```

```
   String temp="";
   while( inputFileRecord != null)
   {
      temp = temp + inputFileRecord;
      System.out.print( "["+ temp + "]" );
      inputFileRecord = inputFile.readLine();
   }

   inputFile.close();
   System.out.println("<---eof");

}//end public static void openReadExistingFile()


public static void addAllValuesInFile() throws IOException
{
   System.out.print("\n At 4 - addAllValuesInFile() : ");
   FileReader      customers = new FileReader("customers.txt");
   BufferedReader  inputFile = new BufferedReader( customers );

   String inputFileRecord;
   inputFileRecord = inputFile.readLine();
   double sumDouble=0;
   int    sumInteger=0;

   while( inputFileRecord != null)
   {
      //converts String to Double
      sumDouble = sumDouble + Double.parseDouble(inputFileRecord);

      //converts String to Double
      sumInteger = sumInteger + Integer.parseInt(inputFileRecord);
      System.out.print( " " + inputFileRecord );
      inputFileRecord = inputFile.readLine();
   }

   inputFile.close();
   System.out.printf(" = Total Double = %.2f  Total Integer = %d",
                     sumDouble, sumInteger);

}//end public static void addAllValuesInFile()


public static int menu()
{
   System.out.print("\n\t*********************************\n\t" +
```

```
                        "*       Enter file mode          *\n\t" +
                        "*   1 - openWriteNewFile         *\n\t" +
                        "*   2 - openWriteExistingFile    *\n\t" +
                        "*   3 - openReadExistingFile     *\n\t" +
                        "*   4 - addAllValuesInFile       *\n\t");

      System.out.print("\n\t  Enter  1, 2, 3, or 0 to exit: ");

      Scanner keyboard = new Scanner(System.in);

      int fileType = keyboard.nextInt();

      return( fileType);
   }//end public static int menu()


   public static void main(String arg[])  throws IOException
   {
      System.out.println("\n Working with files - basic");

      int fileType = menu();

      while( fileType != 0 )
      {
         switch( fileType )
         {
            case 0 : break;
            case 1 : openWriteNewFile(); break;
            case 2 : openWriteExistingFile(); break;
            case 3 : openReadExistingFile(); break;
            case 4 : addAllValuesInFile(); break;
            default : System.out.println( "\tPlease Enter " +
                                          "1, 2, 3, or 0 to exit");
         }
         fileType = menu();

      }//end while( fileType != 0 )

      System.out.println("\n\t End Of Program");

   }//end public static void main(String arg[])

}//end public class filesBasicClass
```

Output:

```
 Working with files - basic

     *********************************
     *       Enter file mode         *
     *   1 - openWriteNewFile        *
     *   2 - openWriteExistingFile   *
     *   3 - openReadExistingFile    *
     *   4 - addAllValuesInFile      *

       Enter  1, 2, 3, or 0 to exit: 1

 At 1 - openWriteNewFile() creating new file and adding:  10 9 8 7 6
5 4 3 2 1 0<---eof

     *********************************
     *       Enter file mode         *
     *   1 - openWriteNewFile        *
     *   2 - openWriteExistingFile   *
     *   3 - openReadExistingFile    *
     *   4 - addAllValuesInFile      *

       Enter  1, 2, 3, or 0 to exit: 2

 At 2 - openWriteExistingFile() Appending data:  10 11 12 13 14
15<---eof

     *********************************
     *       Enter file mode         *
     *   1 - openWriteNewFile        *
     *   2 - openWriteExistingFile   *
     *   3 - openReadExistingFile    *
     *   4 - addAllValuesInFile      *

       Enter  1, 2, 3, or 0 to exit: 3

 At 3 - openReadExistingFile() : [10][109][1098][10987][109876]
[1098765][10987654][109876543][1098765432][10987654321][109876543210]
[10987654321010][109876543210101112]
[109876543210101112131415]<---eof

     *********************************
     *       Enter file mode         *
     *   1 - openWriteNewFile        *
     *   2 - openWriteExistingFile   *
     *   3 - openReadExistingFile    *
```

```
     *   4 - addAllValuesInFile      *

       Enter  1, 2, 3, or 0 to exit: 4

 At 4 - addAllValuesInFile() :  10 9 8 7 6 5 4 3 2 1 0 10 11 12 13 14
15 = Total Double = 130.00  Total Integer = 130
     ********************************
     *       Enter file mode        *
     *   1 - openWriteNewFile       *
     *   2 - openWriteExistingFile  *
     *   3 - openReadExistingFile   *
     *   4 - addAllValuesInFile     *

       Enter  1, 2, 3, or 0 to exit:
```

Text Files using the File Class

Java uses the File class in java.io.File to obtain information about files and directories. The File class does not create files or allows us to make input or obtain output to/from files. Here are some of the methods that we can use with the File class:

```
/**
Author  : Michael Robinson
Program : flesFile.java
Purpose : Checks the status of a  file

Updated : Dec 15, 2011

*/
import java.io.*;

public class filesFile
{

   public static void processFile() throws IOException
   {
      File myFile = new File("1200.dna");

      System.out.print( myFile.getName() );

      if( !myFile.exists() )
      {
         System.out.println( " Does NOT exist" + "\n"  );
      }
      else
      {
         System.out.println( " exists" + "\n" );
         System.out.println( "canExecute()       = " + myFile.canExecute() );
         System.out.println( "canRead()          = " + myFile.canRead() );
         System.out.println( "canWrite()         = " + myFile.canWrite() );
```

```
         System.out.println( "createNewFile()    = " + myFile.createNewFile() );
         //System.out.println( "delete()           = " + myFile.delete() );
         System.out.println( "exists()           = " + myFile.exists() );
         System.out.println( "getClass()         = " + myFile.getClass() );
         System.out.println( "getFreeSpace()     = " + myFile.getFreeSpace() );
         System.out.println( "getName()          = " + myFile.getName() );
         System.out.println( "getParent()        = " + myFile.getParent() );
         System.out.println( "getParentFile()    = " + myFile.getParentFile() );
         System.out.println( "getPath()          = " + myFile.getPath() );
         System.out.println( "getTotalSpace()    = " + myFile.getTotalSpace() );
         System.out.println( "getAbsoluteFile()  = " + myFile.getAbsoluteFile() );
         System.out.println( "getAbsolutePath()  = " + myFile.getAbsolutePath() );
      }

   }//end public static void processFile()


   public static void main(String arg[]) throws IOException
   {
      processFile();
   }//end public static void main(String arg[])

}//end public class filesFile
```

The line

```
         File myFile = new File("1200.dna");
```

checks the current directory for the file called 1200.dna, if it does not exist the program displays “” and terminates, else the program will process the following methods of the File class, applied to the 1200.dna file.

```
         System.out.println( " exists" + "\n" );
         System.out.println( "canExecute()       = " + myFile.canExecute() );
         System.out.println( "canRead()          = " + myFile.canRead() );
         System.out.println( "canWrite()         = " + myFile.canWrite() );
         System.out.println( "createNewFile()    = " + myFile.createNewFile() );
         //System.out.println( "delete()           = " + myFile.delete() );
         System.out.println( "exists()           = " + myFile.exists() );
         System.out.println( "getClass()         = " + myFile.getClass() );
         System.out.println( "getFreeSpace()     = " + myFile.getFreeSpace() );
         System.out.println( "getName()          = " + myFile.getName() );
         System.out.println( "getParent()        = " + myFile.getParent() );
         System.out.println( "getParentFile()    = " + myFile.getParentFile() );
         System.out.println( "getPath()          = " + myFile.getPath() );
         System.out.println( "getTotalSpace()    = " + myFile.getTotalSpace() );
         System.out.println( "getAbsoluteFile()  = " + myFile.getAbsoluteFile() );
         System.out.println( "getAbsolutePath()  = " + myFile.getAbsolutePath() );
```

Notice that I placed remarks // at this line

```
         //System.out.println( "delete()           = " + myFile.delete() );
```

This line will delete the 1200.dna file if the remarks // are removed. There are more methods than these in the File class, please experiment using the other methods. This File class is very useful.

This is the output:

```
1200.dna exists

canExecute()       = false
canRead()          = true
canWrite()         = true
createNewFile()    = false
exists()           = true
getClass()         = class java.io.File
getFreeSpace()     = 67359117312
getName()          = 1200.dna
getParent()        = null
getParentFile()    = null
getPath()          = 1200.dna
getTotalSpace()    = 483398987776
getAbsoluteFile()  = /home/michael/java/JavaPrograms/filesFile/1200.dna
getAbsolutePath()  = /home/michael/java/JavaPrograms/filesFile/1200.dna
```

Exceptions Handling

In Java errors are called Exceptions. There are a large amount of classes that handle errors, allowing us to maintain control of the program. For a detailed explanation of Exceptions see Appendix IV.

To handle an Exception, or control an error, Java requires the following:

- To import the java.util.* package

```
import java.util.*;
```

- To write all the code we want tested inside the try section

```
try
{

}
```

- To tell Java what to do in the catch section, when it finds an error in the code inside the try section.

```
catch(Exception e) //catches ALL errors, can print e
{

}
```

As we see in the catch section we find (Exception e). This is the default Exception handler which can handle any error, Java writes the error inside the variable e, which can be printed.

Finally, the optional finally section can be used for any purpose.

```
   finally
   {
        System.out.println(" At finally. The try catch terminates here\n");
   }
```

In the following program we will see a simple implementation. In the following sections in this chapter you will find additions Exception implementations.

```
import java.util.*;

public class exceptions
{
   public static void main(String arg[])
   {
      Scanner input = new Scanner(System.in);
      int x = 1; //to handle the do while
      int fn=0;
      int sn=0;

      do
      {
         try
         {
            System.out.println(" Enter first number : ");
            fn = input.nextInt();

            System.out.println(" Enter second number : ");
            sn = input.nextInt();

            int result = fn/sn; //if divide by zero occurs, java jumps to catch

            x = 2; //if divide by zero occurs this line does NOT execute
         }
         catch(Exception e) //catches ALL errors, can print e
         {
            System.out.println(" Division by zero NOT possible");
            System.out.println(" Error = " + e.getMessage() );
         }
         finally
         {
            System.out.println(" At finally. The try catch terminates here\n");
         }

      } while( x == 1); //go back to the beginning of do loop

      float result2 = (float)fn/sn; //only if NOT divide by zero error
```

```
      System.out.printf( " %d / %d = %.2f", fn, sn, result2 );

      System.out.println("\nEnd Exception e sample");
```

```
This is the output:
 Enter first number : 40
 Enter second number : 0
 Division by zero NOT possible
 Error = / by zero
 At finally. The try catch terminates here

 Enter first number : 40
 Enter second number : 2
 At finally. The try catch terminates here

 40 / 2 = 20.00
End Exception e sample
```

We must have one try section, and as many catch sections as needed. The finally section is optional.

Text Files using the Formatter Class

In Java there are several ways to create files, using the Formatter class is probably the easiest and fastest way to create a file, Formatter also allows to write data into the file. However make sure this file does NOT exist otherwise it will overwrite it (delete and re-create).

```
/*
Author  : Michael Robinson
Program : filesFormatter.java
Purpose : This program uses the Formatter class
          this is a very quick way
          to create the file called fileone then we
          write data into into and finally
          close it.

Updated : Jan 29, 2011
*/

import java.util.*;

public class filesFormatter
{
   private static Formatter fileName;

   public static void createAfile()
   {
```

```
      try
      {
         //make sure this file does NOT exist
         //otherwise it will overwrite it (delete and re-create)
         fileName = new Formatter("fileOne");
         System.out.println("I have created fileOne");
      }
      catch(Exception e)
      {
         System.out.println("fileOne was not created ERROR!!");
      }

   }//end public static void createAfile()


   public static void writeDataToFile( String fn, String ln, String major)
   {
      try
      {
         //exactly as using printf
         fileName.format("%s %s %s\n", fn, ln, major);
         System.out.println("Inserted record : " + fn + " " + ln + " " + major );
      }
      catch(Exception e)
      {
         System.out.println("fileOne was not created ERROR!!");
      }

   }//end public static void writeDataToFile()


   public static void closeFile()
   {
      fileName.close();
      System.out.println( "File has been closed");

   }//end public static void closeFile()


   public static void main(String arg[])
   {
      createAfile();

      writeDataToFile( "Joe", "Smith", "CS" );
      writeDataToFile( "Tom", "Richards", "IT" );
      writeDataToFile( "Daniel", "Thomason", "Math" );
      writeDataToFile( "Robert", "Lambert", "EE" );
      writeDataToFile( "Miguel", "Gonzalez", "CS" );

      closeFile();

   }//end   public static void main(String arg[])

}//end public class filesFormatter
```

As we can see to create a file using the Formatter class all we need to is create a final Formatter object

```
final Formatter fileName;
```

and then give it the name of the file that will be created in the secondary storage (hard disk/cd/tape/flash/etc):

```
fileName = new Formatter("fileOne");
```

When we write data to the file we use the object name for the file (fileName) and its method format. This method is used exactly in the same manner we we the printf method as follows:

```
fileName.format("%s %s %s\n", fn, ln, major);
```

This is the output:

```
I have created fileOne
Inserted record : Joe Smith CS
Inserted record : Tom Richards IT
Inserted record : Daniel Thomason Math
Inserted record : Robert Lambert EE
Inserted record : Miguel Gonzalez CS
File has been closed
```

This section can be used in combination with any other files classes, for instance, now we are going to use the Scanner class to read the data in the file we just created.

Text Files using the Scanner Class

The Scanner class is usually used to obtain data from the keyboard, here we will see we can also use it to open text files and process them.

```
/*
Author  : Michael Robinson
Program : filesScanner.java
Purpose : This program uses the Scanner class
          this is a very quick way
          to read data from a text file

Updated : December 21, 2011
*/

import java.io.*;
import java.util.*;
```

```
public class filesScanner
{
   private static Scanner inputFile;

   public static void openFile()
   {
      try
      {
         inputFile = new Scanner( new File("fileOne") );
      }
      catch(Exception e)
      {
         System.out.println( "Could not open file " );
      }

   }//end public static void openFile()


   public static void readFromFile()
   {
      int record = 0;
      while( inputFile.hasNext())
      {
         record++;
         String fn    = inputFile.next();
         String ln    = inputFile.next();
         String major = inputFile.next();

         System.out.printf( " %d %s %s %s\n", record, fn, ln, major );
      }
   }//end public static void readFromFile()


   public static void closeFile()
   {
      inputFile.close();
   }//end public static void closeFile()


   public static void main(String arg[])
   {
      openFile();
      readFromFile();
      closeFile();
   }//end public static void main(String arg[])


}//end public class filesScanner
```

This is the output:

```
 1 Joe Smith CS
 2 Tom Richards IT
 3 Daniel Thomason Math
 4 Robert Lambert EE
 5 Miguel Gonzalez CS
```

The Scanner class has many methods and we will see some of them in different sections.

Binary Files

Binary files can only be seen by binary editors and usually they are compiled programs that are executed.

This is how Java handles Binary Files:

```
/*
Author  : Michael Robinson
Program : FilesBinary.java
Purpose : This program
          creates a binary files
          writes data to it
          closes file and then
          opens the binary file,
          reads and displays the contents.

Updated : Jan 29, 2011
*/

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class filesBinary
{
   public static void createBinaryFile( int numbers[] ) throws IOException
   {
      //An array to be written to a file
      //int[] numbers = { 2, 4, 6, 8, 10, 12, 14 };

      //Create the binary output objects
      //if file exists it appends data to it
      //else creates file, done by true parameter.
      FileOutputStream outputBinary = new FileOutputStream("numbers.dat", true);
      DataOutputStream outputBinFile = new DataOutputStream(outputBinary);

      System.out.println("Writing the numbers to the file...");

      // Write the array elements to the file.
      for (int i = 0; i < numbers.length; i++)
      {
         outputBinFile.writeInt( numbers[i] );
         System.out.print( numbers[i] + " " );
      }
      //outputBinFile.writeChars("\n");
```

```
   System.out.println("\nDone.");

   // Close the file.
   outputBinFile.close();

}//end public static void createBinaryFile()


public static void openReadBinaryFile( int numbersLength ) throws IOException
{
   int number; // A number read from the file
   boolean endOfFile = false;     // EOF flag

   // Create the binary file input objects.
   FileInputStream inputBinary  =  new FileInputStream("numbers.dat");
   DataInputStream inputBFile   =  new DataInputStream(inputBinary);

   System.out.println("\nReading numbers from Binary file:");

   // Read the contents of the file.
   while (!endOfFile)
   {
      try
      {
         for(int x=0; x < numbersLength; x++)
         {
            number = inputBFile.readInt();
            System.out.print( number );
         }
         System.out.println();
      }
      catch (EOFException e)
      {
         endOfFile = true;
      }
   }

   System.out.println("\nDone.");

   // Close the file.
   inputBFile.close();

}//end public static void openReadBinaryFile() throws IOException


public static void main(String[] args) throws IOException
{
   // An array to write to the file
   int[] numbers = { 2, 4, 6, 8, 10, 12, 14 };

   createBinaryFile( numbers );
   openReadBinaryFile( numbers.length );

}//end public static void main(String[] args) throws IOException
```

```
}//end public class filesBinary

This is the output:
Writing the numbers to the file...
2 4 6 8 10 12 14
Done.

Reading numbers from Binary file:
2468101214
2468101214
2468101214
2468101214
2468101214
2468101214
2468101214
Done.
```

Printing numbers using decimal format

Now that we have learned how to work with files, we need to learn to print numeric data in with dollar signs, numbers separated with commas and periods, and multiple decimal numbers. For this Java has a class called DecimalFormat.

```
/*
Author  : Michael Robinson
Program : decimalFormatClass.java
Purpose : How to print numbers with decimal formats
Updated : Jan 29, 2011
*/

import java.text.DecimalFormat;
import java.util.Scanner;

public class decimalFormatClass
{
   public static void main(String arg[])
   {
      int x;
      double amount = 100;

      DecimalFormat moneyFormat = new DecimalFormat("$ #,###.000");

      for(x=0; x < 5; x++)
      {
         amount = amount + (.005 * x);
         System.out.println("\nOriginal : " + amount);
         System.out.println("Formatted: US " +
```

```
                              moneyFormat.format(amount) );
      }//end for loop

   }//end public static void main(String arg[])

}//end class decimalFormatClass
```

This is the output:

```
Original : 100.0
Formatted: US $ 100.000

Original : 100.005
Formatted: US $ 100.005

Original : 100.015
Formatted: US $ 100.015

Original : 100.03
Formatted: US $ 100.030

Original : 100.05
Formatted: US $ 100.050
```

First we need to import the class:

```
                  import java.text.DecimalFormat;
```

To create the numerical format to be used we need:

```
   DecimalFormat moneyFormat = new DecimalFormat("$ #,###.000");
```

DecimalFormat is the class to be used
moneyFormat is the name of the object we are creating
$ #,###.000 is the format we want. This means:
- place a $ leaving a space before the number
- #,###.000 write the number using this format, using commas as needed, and a period for decimals If there are no decimal place up to three zeros.

To print use the object moneyFormat.format(amount) as follows:

```
         System.out.println("Formatted: US " +
                              moneyFormat.format(amount) );
```

Modifications of this format are allowed, using different currency symbols and amount of whole numbers and/or decimals.

//end **Chapter Nine : Data Structures – Files**

Chapter Ten : Constructors and
Variable-Length Argument List in Methods

What is a Constructor

Up to now we have been using classes provided to us by Java, now we are going to learn how to create our own classes using constructors. Constructors create objects, also called an instance of a class. I also refer to them as clones.

These are examples of some of the Java classes we have used so far.

```
Class              Object            Data for the Constructor
String             str         = new String("Hello");
File               myFile      = new File("1200.dna");
Scanner            keyBoard    = new Scanner(System.in);
FileWriter         customers   = new FileWriter("customers.txt");
PrintWriter        output      = new PrintWriter( customers );
FileReader         customers   = new FileReader("customers.txt");
BufferedReader     inputFile   = new BufferedReader( customers );
DecimalFormat      moneyFormat = new DecimalFormat("$ #,###.000");
ArrayList<String>  arrayList   = new ArrayList<String>();
```

Just like in the above examples we can create our own classes, but in addition we can create classes that accept multiple parameters, for instance, if we create a class called building, which calculates the square area of a room, we can create it so that it can accept different amount, and data types of variables, e.g:

```
building room0 = new building();           //no variables passed
building room1 = new building(40, 20);     //pass two ints
building room2 = new building(40.5, 20);   //pass one double one int
building room3 = new building(40, 20.7);   //pass one int one double
building room4 = new building(40.5, 19.5); //pass two doubles
```

Simple Constructors

Simple Constructors have four sections:

- = The global variables
- = The constructors
- = The mutator or setter methods
  Mutator methods are usually named "***set***Something", and are used to receive variables from the calling program and assign them to the global variables.
- = The accessors or getter methods
  Accessor methods are usually named "***get***Something", and are used to

return the global values to the calling program.

So before we can create objects/instance of a class, we need to create the class, the following program is a simple example:

```
/**
Author  : Michael Robinson modified from book
Program : building.java
Purpose : To present Classes and Objects
          - an object is an instance of a class, a clone of a class
          - an object contains methods and data called object's fields.
          - objects are not stand alone programs but can be re-used like methods.
          - the data is created from variables and more powerful data structures


Updated : Dec 23, 2011

*/

public class building
{
      private double length;
      private double width;

      /**
         Constructors
         @param the first constructor does not accept variables
         @param the other constructors accept different data types
         @param len is the length of the rectangle.
         @param w   is the width of the rectangle.
      */

      public building()
      {
         length = 15;
         width = 15;
      }

      public building(int len, int w)
      {
         length = len;
         width = w;
      }

      public building(double len, int w)
      {
         length = len;
         width = w;
      }

      public building(int len, double w)
      {
         length = len;
```

```
   width = w;
}

public building(double len, double w)
{
   length = len;
   width = w;
}


/***
   The setLength method stores a value in the length field.
   @param len The value to store in length.
*/

//Mutators (sets) ************************************
public void setLength(double len)
{
   length = len;
}

/**
   The setWidth method stores a value in the width field.
   @param w The value to store in width.
*/

public void setWidth(double w)
{
   width = w;
}


//Accessors (gets) ***********************************
/**
   The getLength method returns a Rectangle object's length.
   @return The value in the length field.
*/

public double getLength()
{
   return length;
}

/**
   The getWidth method returns a Rectangle object's width.
   @return The value in the width field.
*/

public double getWidth()
{
   return width;
}

/**
   The getArea method returns a Rectangle object's area.
   @return The product of length times width.
```

```
      */

      public double getArea()
      {
         return length * width;
      }

}//end public class building
```

The above program has the following constructors:

```
      public building()

      public building(int len, int w)

      public building(double len, int w)

      public building(int len, double w)

      public building(double len, double w)
```

As we can see, we have created five different types of objects, each receiving different data types sequences.

In addition to standard Java methods such as: equals(), getClass(), hashCode(), notify(), etc, the above constructors will inherit/have access to the following methods:

```
      //Mutators (sets) ************************************
      public void setLength(double len)
      public void setWidth(double w)

      //Accessors (gets) ***********************************
      public double getLength()
      public double getWidth()
      public double getArea()
```

We can have any amount of constructors and methods in each class.

Now we need a program that can make use of our “building” class, here is a small example:

```
/**
Author  : Michael Robinson
Program : buildingSamples.java
Purpose : To present Classes and Objects
          This program create five different objects from
          out own constructor, each object corresponds to
          a different constructor that received different
          data types.

Updated : December 23, 2011
```

```
*/

public class buildingSamples
{
  //this section creates five global objects with different data types
  public static building room0 = new building();         //no variables passed
  public static building room1 = new building(40, 20);   //pass two ints
  public static building room2 = new building(40.5, 20); //pass one double one int
  public static building room3 = new building(40, 20.7); //pass one int one double
  public static building room4 = new building(40.5, 19.5); //pass two doubles


   public static void doAccessors()
   {
      //these sections obtain the len, width and area of all rooms (objects)
      System.out.println( "\nRoom0" );
      //Accessors (gets) ************************************
      System.out.printf("Room0's length is %.2f\n", room0.getLength() );
      System.out.printf("Room0's width  is %.2f\n", room0.getWidth() );
      System.out.printf("Room0's area   is %.2f\n", room0.getArea() );


      System.out.println( "\nRoom1" );
      //Accessors (gets) ************************************
      System.out.printf("Room1's length is %.2f\n", room1.getLength() );
      System.out.printf("Room1's width  is %.2f\n", room1.getWidth() );
      System.out.printf("Room1's area   is %.2f\n", room1.getArea() );


      System.out.println( "\nRoom2" );
      //Accessors (gets) ************************************
      System.out.printf("Room2's length is %.2f\n", room2.getLength() );
      System.out.printf("Room2's width  is %.2f\n", room2.getWidth() );
      System.out.printf("Room2's area   is %.2f\n", room2.getArea() );


      System.out.println( "\nRoom3" );
      //Accessors (gets) ************************************
      System.out.printf("Room3's length is %.2f\n", room3.getLength() );
      System.out.printf("Room3's width  is %.2f\n", room3.getWidth() );
      System.out.printf("Room3's area   is %.2f\n", room3.getArea() );


      System.out.println( "\nRoom4" );
      //Accessors (gets) ************************************
      System.out.printf("Room4's length is %.2f\n", room4.getLength() );
      System.out.printf("Room4's width  is %.2f\n", room4.getWidth() );
      System.out.printf("Room4's area   is %.2f\n", room4.getArea() );

   }//end public static void doAccessors()


   public static void doMutators()
   {
      System.out.println( "\nChanging measurements in the new building..." );
```

```
      //now we are going to change measurements in each room (object)
      //mutators (setters) ************************************

      room1.setLength( 10 );
      room1.setWidth( 4 );

      room2.setLength( 10.9 );
      room2.setWidth( 7 );

      room3.setLength( 8 );
      room3.setWidth( 14.4 );

      room4.setLength( 10.2 );
      room4.setWidth( 40.1 );

   }//end public static void doMutators()



   public static void main(String arg[])
   {
      System.out.println( "Finding measurements in the new building" );

      doAccessors();

      doMutators();

      doAccessors();

   }//end public static void main(String arg[])

}//end public class buildingSamples
```

This is the output:

```
Finding measurements in the new building

Room0
Room0's length is 15.00
Room0's width  is 15.00
Room0's area   is 225.00

Room1
Room1's length is 40.00
Room1's width  is 20.00
Room1's area   is 800.00

Room2
Room2's length is 40.50
Room2's width  is 20.00
Room2's area   is 810.00

Room3
Room3's length is 40.00
```

```
Room3's width  is 20.70
Room3's area   is 828.00

Room4
Room4's length is 40.50
Room4's width  is 19.50
Room4's area   is 789.75

Changing measurements in the new building...

Room0
Room0's length is 15.00
Room0's width  is 15.00
Room0's area   is 225.00

Room1
Room1's length is 10.00
Room1's width  is 4.00
Room1's area   is 40.00

Room2
Room2's length is 10.90
Room2's width  is 7.00
Room2's area   is 76.30

Room3
Room3's length is 8.00
Room3's width  is 14.40
Room3's area   is 115.20

Room4
Room4's length is 10.20
Room4's width  is 40.10
Room4's area   is 409.02
```

As we can see the constructor “building.java” that we created above can be used by any program at any location. This class actually has five build-in constructors accepting different sets of data types. We can also build classes that will accept multiple amounts of data types. Think of the class “System.out.prinf”, how many variations does it have?, I do not know, but it would be interesting to study it's source code.

Notice that the constructor “building.java” does not have the main method, however the calling program “buildingSamples” being a regular program does have the main method.

What is a Variable-Length Argument Lists in Methods

If we need to process records from a file where each record could have different amounts of fields or items e.i.

```
2,74,6,8,5,60
```

```
2,74
2,74,6,8
2,74,6,8,5
2,74,6
```

and we need to pass each record to a method that adds those items, so that we can then obtain an average, with our current knowledge we will have to create a method for each record based on its amount of fields. However Java has implemented a way where we only need one method regardless of the amount of items we are passing. This is called “Variable-Length Argument Lists”.

The data these methods receive must be of the same type, it could ints, floats, String, etc, but each group must be of the same data type.

The following program shows in detail how to implement this very useful tool.

Notice that the variable that is accepted by the method is automatically converted to a single dimension array, so to access each value follow the single dimension array rules. Also notice that to access each value we can use regular for loops, while loops and the enhanced for loop.

```
/**
Author  : Michael Robinson
Program : varLenArgumentsClass.java
Purpose : To call a method passing VARIABLE AMOUNT OF PARAMETERS
          Notice that the variable that is accepted by the method
          is automatically converted to an array, so that to
          access each value follow the single dimension array
          rules.
          To access each value we can use regular for loops, while
          loops or the enhanced for loop.

Updated : April 6th, 2012

*/


public class varLenArgumentsClass
{
   //anytime we need to accept variables and we do not know how many are going to
   //be passed we use ( int... variableName ), the three .'s mean that it could be
   //any amount.
   public static int getIntAverage( int... numbers )
   {
      //accepts unknown amount of ints
      int total = 0;
      int counter = 0;
```

```
   for( int x : numbers ) //using the enhanced for loop
   {
      System.out.print( x );
      if( counter < numbers.length - 1 )
      {
         System.out.print( " + " );
         counter++;
      }

      total += x;
   }
   System.out.printf( " = %d ", total );
   return ( total / numbers.length );
}


public static float getFloatAverage( float... numbers )
{
   //accepts unknown amount of floats
   float total = 0;
   int counter = 0;

   for( int x = 0; x < numbers.length; x++ ) //using the regular for loop
   {
      System.out.printf( "%.2f" , numbers[x] );
      if( counter < numbers.length - 1 )
      {
         System.out.print( " + " );
         counter++;
      }
      total+=x;
   }
   System.out.printf( " = %.2f ", total );
   return ( total / numbers.length );
}


public static void printNames( String... theStrings )
{
   //accepts unknown amount of Strings
   int total = 0;
   int x = 0;

   while( x < theStrings.length ) //using a while loop
   {
      System.out.print( theStrings[x] + " " );
      x++;
   }
   System.out.println( "" );
}


public static void main( String arg[] )
{
   System.out.println( "Processing ints" );
```

```
      System.out.println( "\t[  The Average is : " +
                                 getIntAverage(2,74,6,8,5,60) + "  ]" );
      System.out.println( "\t[  The Average is : " +
                                 getIntAverage(2,74,6,8,5) + "  ]" );
      System.out.println( "\t\t[  The Average is : " +
                                  getIntAverage(2,74,6,8) + "  ]" );
      System.out.println( "\t\t[  The Average is : " + getIntAverage(2,74,6)
                                  + "  ]" );
      System.out.println( "\t\t\t[  The Average is : " + getIntAverage(2,74)
                                  + "  ]" );

      System.out.println( "\nProcessing Strings" );
      printNames( "Pilar", "Yeesenia", "RoseMarie", "Mark", "Daniel", "Paula",
                  "Camila", "Daniela", "Michael" );
      printNames( "Pilar", "Yeesenia", "RoseMarie", "Mark", "Daniel", "Paula",
                  "Camila", "Daniela" );
      printNames( "Pilar", "Yeesenia", "RoseMarie", "Mark", "Daniel", "Paula",
                  "Camila" );
      printNames( "Pilar", "Yeesenia", "RoseMarie", "Mark", "Daniel", "Paula" );
      printNames( "Pilar", "Yeesenia", "RoseMarie", "Mark", "Daniel"  );


      System.out.println( "\n\nProcessing floats" );
      System.out.printf( "\t[  The Average is : %.2f   ]\n",
                               getFloatAverage(2,74,6,8,5,60)  );
      System.out.printf( "\t\t[  The Average is : %.2f   ]\n",
                                getFloatAverage(2,74,6,8,5)  );
      System.out.printf( "\t\t\t[  The Average is : %.2f   ]\n",
                                 getFloatAverage(2,74,6,8)  );
      System.out.printf( "\t\t\t\t[  The Average is : %.2f   ]\n",
                                   getFloatAverage(2,74,6)  );
      System.out.printf( "\t\t\t\t\t[  The Average is : %.2f   ]\n",
                                     getFloatAverage(2,74)  );
   }

}
```

The methods:

```
      public static int getIntAverage( int... numbers )

      public static float getFloatAverage( float... numbers )

      public static void printNames( String... theStrings )
```

accept unknown amount of fields therefore these methods can be re-used from any place, and as many times as needed, accepting any amount of items. The only resctriction is that the data being passed and received must be of the same data type. e.i. int, float, char, String, arrays, wrappers, etc.

This works because the data being received is accepted by a variable which is actually a single dimesion array. I know this could be

confusing because we are used to declare arrays with the [] brackets and in this form the [] brackets are missing. Since these variables are a single dimensional array without any size assigned to it, it can accept any amount of items.

To process these data items we can use any of the methods use to process arrays such as for, while and enhanced for loops.

```
Output:
Processing ints
2 + 74 + 6 + 8 + 5 + 60 = 155  [  The Average is : 25  ]
2 + 74 + 6 + 8 + 5 = 95        [  The Average is : 19  ]
2 + 74 + 6 + 8 = 90            [  The Average is : 22  ]
2 + 74 + 6 = 82                [  The Average is : 27  ]
2 + 74 = 76                    [  The Average is : 38  ]

Processing Strings
Pilar Yeesenia RoseMarie Mark Daniel Paula Camila Daniela Michael
Pilar Yeesenia RoseMarie Mark Daniel Paula Camila Daniela
Pilar Yeesenia RoseMarie Mark Daniel Paula Camila
Pilar Yeesenia RoseMarie Mark Daniel Paula
Pilar Yeesenia RoseMarie Mark Daniel


Processing floats
2.00 + 74.00 + 6.00 + 8.00 + 5.00 + 60.00 = 15.00    [  The Average is : 2.50   ]
2.00 + 74.00 + 6.00 + 8.00 + 5.00 = 10.00            [  The Average is : 2.00   ]
2.00 + 74.00 + 6.00 + 8.00 = 6.00                    [  The Average is : 1.50   ]
2.00 + 74.00 + 6.00 = 3.00                           [  The Average is : 1.00   ]
2.00 + 74.00 = 1.00                                  [  The Average is : 0.50   ]
```

What is the meaning of main(String arg[])

Since the beginning we have been using the main method:

```
public static basic main( String arg[] )
```

but we have not mentioned what is the purpose of the (String arg[]) in the main method. Well it is similar to the (int... variable) in the Variable-Length Argument Lists methods.

The String arg[] inside the main method is simply the single dimensional array arg of String data type which will accept any amount of items from another program or the command line. For instance you have a program written in C which produced multiple output values and it needs the Java program to accept and use these variables in a specific process. So call the Java program passing these variables, and the Java program thru the main method will accept them.

The following is an example if you are using the command line with the following Java program:

```
public class mainString
{
      public static void main(String arg[])
      {
            for(int x = 0; x < arg.length; x++)
            {
                  System.out.print( " [" + arg[x] + "]" );
            }
            System.out.println();
      }
}
```

This is the process:

This line compiles the mainString.java Java program as follows:

```
HOST:~/java/JavaPrograms/mainString/src$   javac mainString.java
```

This line runs the compiled Java passing Hello World to it, as follows:

```
HOST:~/java/JavaPrograms/mainString/src$   java mainString Hello World
```

This is the output:

```
 [Hello] [World]
```

Remember since String arg[] is an array of String data type and an undefined length, it will accept just about any input.

//end **Chapter Ten : Constructors**

Chapter Eleven : GUI Applications - Beginning

What is GUI (Graphical User Interface)

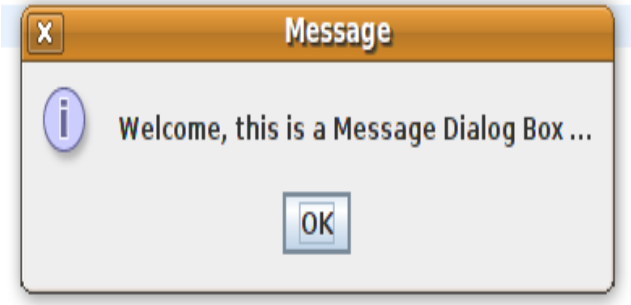
In Computer Science the term GUI, usually pronounced gooey, is how users communicate with electronic machines such as monitors, cell phones, tablets, etc. using graphical images instead of text commands. For the user, using GUI usually makes the comunication easier, simply point and click. The disadvantages are that the communication is limited to the GUI's options presented to the user. Using text commands to communicate with electronic machines allows us to have total control of the machine.

What are Dialog Boxes

In Graphical User Interface (GUI) programming, among other things we use Dialog Boxes to enable communication between the user and the program. In this chapter we are going to use two types of Dialog Boxes, Message Dialog Boxes and Input Dialog Boxes.
Dialog Boxes communicate information to the users, requesting a response from the users.

Message Dialog Boxes

These boxes usually contain a system icon, one or more buttons, and a short specific message.



Input Dialog Boxes

Dialog boxes are used to interact with the user and retrieve information.

These boxes usually contain a system icon, multiple buttons, a short specific message, and a section where the user can input information.

These boxes stops the users from accessing other sections of the program before replying to the current Dialog Box.

All the information received from the Input Dialo Boxes is of String data type, regarless of what data type was written in the Dialog Box. Java provides the following methods to convert the data entered from String type to the intended data type.

If the data entered is assgined to a variable called temp, it can be casted/converted as follows:

```
Integer response = Integer.parseInt(temp);
Double  response = Double.parseDouble(temp);
Float   response = Float.parseFloat(temp);
Long    response = Long.parseLong(temp);
Short   response = Short.parseShort(temp);
```

The following program will shows us how to create Dialog Boxes.

```
/*
Author  : Michael Robinson
Program : gui_1_dialogBoxes.java
Purpose : To present GUI dialog boxes
          - Message Dialog Boxes
          - Input Dialog Boxes

Notes   : JOptionPane ALWAYS returns the input as STRING regardless of what was
inputed

Updated : July 5th, 2012
*/


import javax.swing.JOptionPane;

public class gui_1_dialogBoxes
{

   public static String doInputBox( String question )
   {
      return( JOptionPane.showInputDialog( question ) );
   }
```

```
public static void doMessageBox( String message )
{
   JOptionPane.showMessageDialog( null, message );
}


public static String swap( Object ... numbers )
{
   String array[] = new String[100];

   int counter = 0;

   String temp = "";

   //accessing the items passed
   for( Object x : numbers ) //using the enhanced for loop
   {
      array[counter] = x.toString();
      counter++;
   }

   if( array[0].compareTo(array[1]) > 0 )
   {
      temp     = array[0];
      array[0] = array[1];
      array[1] = temp;
   }

   if( array[1].compareTo(array[2]) > 0 )
   {
      temp     = array[1];
      array[1] = array[2];
      array[2] = temp;
   }


   if( array[0].compareTo(array[1]) > 0 )
   {
      temp     = array[0];
      array[0] = array[1];
      array[1] = temp;
   }


   //JOptionPane ALWAYS returns the input as STRING regardless of what was
   //inputed
   return ( "Sorted :  " + array[0] + "  " +  array[1] + "  " + array[2] );

}//end   public static String swap( int ... numbers )



public static void main(String arg[])
{
   doMessageBox("Welcome, this is a Message Dialog Box ...");
```

```
      String fName = doInputBox( "Enter your First Name" );

      String mName = doInputBox( "Enter your Middle Name" );

      String lName = doInputBox( "Enter your Last Name" );

      doMessageBox("Nice to meet you\n" + fName + " " + mName + " " + lName +
                   "\nNow let's do some Sorting");

      String fNum = ( doInputBox( "Enter First Object" ) );

      String sNum = ( doInputBox( "Enter Second Object" ) );

      String tNum = ( doInputBox( "Enter Third Object" ) );


      doMessageBox("Objects Swap " + fName + "\n" + "Before :  " + fNum + "  " +
                    sNum + "  " + tNum +"\n" + swap( fNum, sNum, tNum ) );


      System.exit(0);  //must use when using JOptionPane b/c it causes an
                       //additional task to run in the VM

   }//end public static void main(String arg[])

}//end     public static void main(String arg[])
```

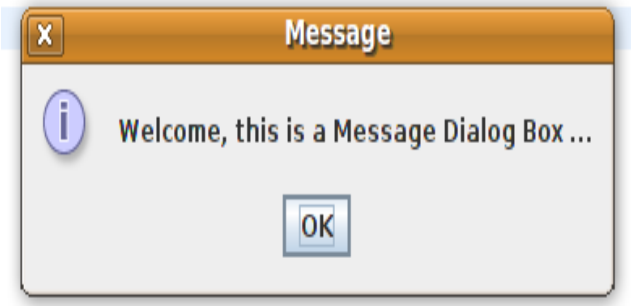
Notice that the line:

```
       public static void doMessageBox( String message )
```

Receives a String in the variable message.

The following line produces the Message Dialog box below

```
        JOptionPane.showMessageDialog( null, message );
```



The line

```
      String fName = doInputBox( "Enter your First Name" );
```

calls the following method passing a String that will appear in an Input Dialog Box.

```
public static String doInputBox( String question )
```

This method creates the Input Dialog Box, request data from the user, accepts it, and returns it to the calling command above, as follows:

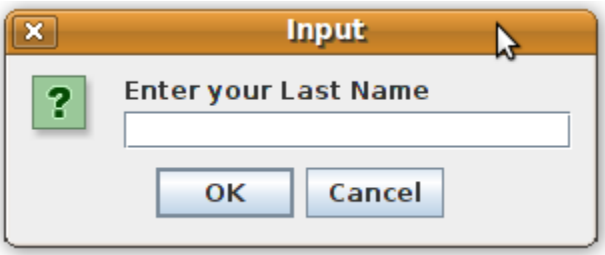
```
return( JOptionPane.showInputDialog( question ) );
```

The following Input Dialog Boxes used in this program are implemented as follows:

```
String fName = doInputBox( "Enter your First Name" );
```


```
String mName = doInputBox( "Enter your Middle Name" );
```


```
String lName = doInputBox( "Enter your Last Name" );
```



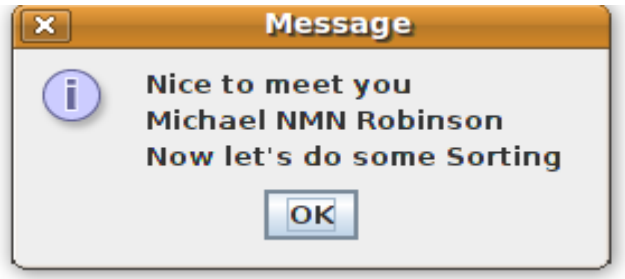
The data inputed by the user is then passed as follows:

```
doMessageBox("Nice to meet you\n" + fName + " " + mName + " " + lName +
             "\nNow let's do some Sorting");
```

to the following method:

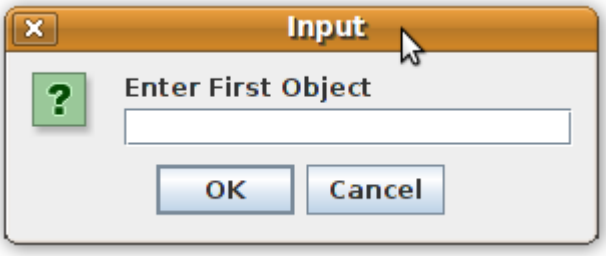
```
public static void doMessageBox( String message )
{
   JOptionPane.showMessageDialog( null, message );
}
```

This method creates a Message Dialog Box displaying the following results:

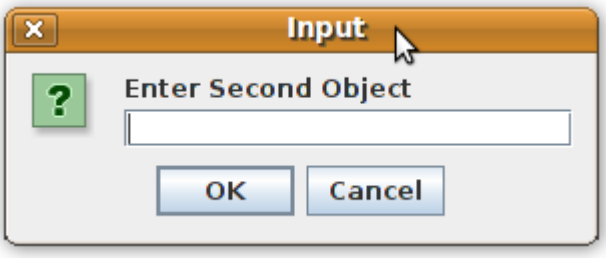


The following Input Dialog Boxes will accept data of Object type using the following command:

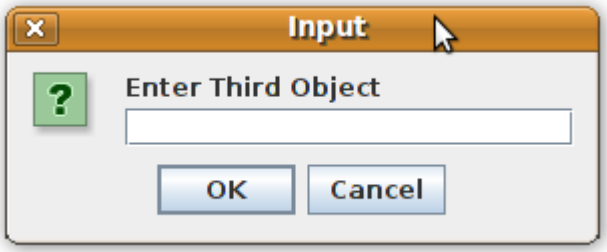
```
String fNum = ( doInputBox( "Enter First Object" ) );
```




```
String sNum = ( doInputBox( "Enter Second Object" ) );
```




```
String tNum = ( doInputBox( "Enter Third Object" ) );
```



The data acquired by these Input Dialog boxes will be send to the following method:

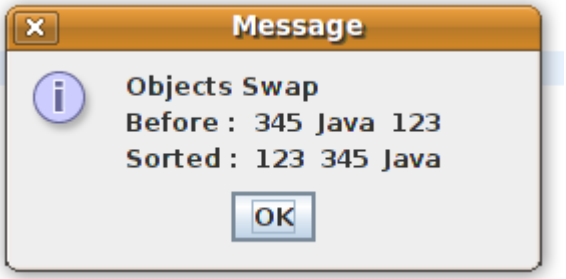
```
public static String swap( Object ... numbers )
```

using the following command:

```
doMessageBox("Objects Swap " + fName + "\n" + "Before :  " + fNum + "  " +
              sNum + "  " + tNum +"\n" + swap( fNum, sNum, tNum ) );
```

The swap method will sort its input in ascending order and return it to the calling command.

The original data and the sorted data will be displayed in a Message Dialog Box.



What is a Frame

In GUI programming, a Frame is a window with a title and a border. GUI applications have at least one frame with button components that close, minimize or maximizes the window.

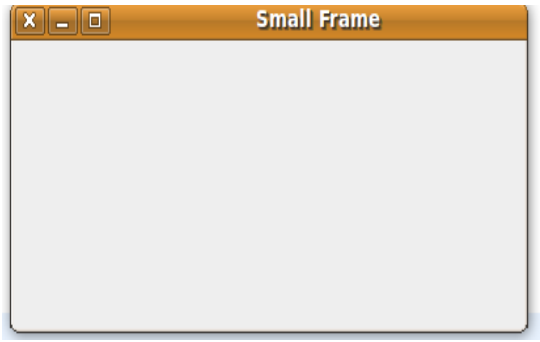
Creating a Frame

Creating a Frame that is not full size is easier than a full size Frame, when we execute a GUI program we expect a full screen. The following program will show how to create both.

Creating a Small Frame

```
/*
Author  : Michael Robinson
Program : guiSmallFrame.java
Purpose : To create a small GUI Frame

Updated : July 5th, 2012
*/


import javax.swing.JFrame;


public class guiSmallFrame
{

      public static void createSmallFrame()
      {
            //create a frame also known as a window
            JFrame frame = new JFrame("Small Frame");
            frame.setVisible(true);
            frame.setSize(400,200);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }//end public static void createSmallFrame()


    public static void main(String arg[])
    {
      createSmallFrame();
    }//end public static void main(String arg[])


}//end public class guiSmallFrame
```

This is the output:



It appears at the top left corner of the monitor.

We first need to **import** javax.swing.JFrame; to be able to access all the JFrame classes.

Using the JFrame class we create an object as follows:

```
JFrame frame = new JFrame("Small Frame");
```

The String inside the JFrame in this example "Small Frame" will appear on the top Bar section of the Frame.

```
frame.setVisible(true);
```

Makes the Frame visible.

```
frame.setSize(400,200);
```

Creates a Frame of size 400 by 200, you can set it to any size.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

When we press the X box on the Bar the Frame will close.

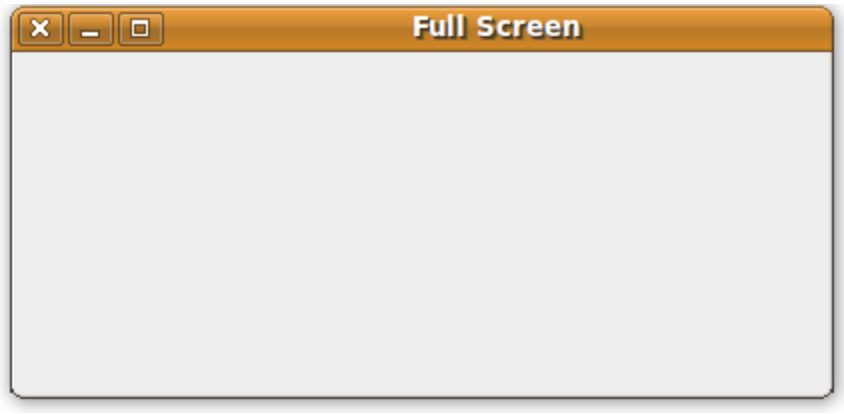
Creating a Full Size Frame

When we execute a GUI program we expect a full screen. The following program will show how to create a one.

```
/*
Author  : Michael Robinson
Program : guiFullScreen.java
Purpose : To create a full size GUI Frame

Updated : July 5th, 2012
*/


import javax.swing.*;
import java.awt.Toolkit;

public class guiFullScreen
{
      private static JFrame frame;

      public static void createGuiFullScreen()
      {
            frame = new JFrame("Full Screen");
            Toolkit tk = Toolkit.getDefaultToolkit();
            int xSize = ((int) tk.getScreenSize().getWidth());
            int ySize = ((int) tk.getScreenSize().getHeight());
            frame.setSize(xSize,ySize);
            frame.setVisible(true);
      }//end      public static void createGuiFullScreen()
```

```
	public static void main(String[] args)
	{
		createGuiFullScreen();

	}//end public static void main(String[] args)

}//end public class GuiFullScreen
```

This is the output:



//end Chapter Eleven : GUI Applications