Introduction

Java is an object oriented programming language developed by Sun Microsystems. Modeled after C++. It was designed to be small, simple and portable across platforms and operating systems both at the source and at the binary level which means that Java programs (applications and applets) can run on any machine that has the Java virtual machine has been installed.

Applets

Java applets are programs that are downloaded from the worldwide web by a web browser and run inside a HTML web page. One needs a Java enabled browser to run applets.

To create an applet you write it in the Java language, compile it using the Java compiler and refer to that applet in your HTML web pages.

While applets are the most popular use of Java you can also write full-fledged general purpose programs the same way you do using C or C++. These programs are referred to as applications.

The Java language was developed at Sun Microsystems in 1991 as part of a research project to develop software for consumer electronics devices – television sets, VCRs, toasters etc. Java's goals at that time were to be small, fast, efficient and easily portable to a wide range of hardware devices. Those same goals made Java an ideal language for distributing executable programs via the Worldwide Web and also a general purpose programming language for developing programs that are easily usable and portable across different platforms.

Reasons for Learning JAVA

One of the most compelling reasons to learn Java is because applets are written in Java. The other reason is that it is suitable for just about any programming task. One of the reasons for this is: -

Java is platform independent

Java programs can be moved easily from one computer system to another and this is one of the most significant advantages Java has over other programming languages. When writing software for the worldwide web, the program must have the ability to run on many different platforms.

Platform Independence

Platform independence means that a program can run on any computer system.

When you compile a program written in C or in most other languages, the compiler translates your program into machine code or processor instructions which are specific to the processor your computer is running. With Java on the other hand, the development environment has two parts: a Java compiler and a Java interpreter. The Java compiler takes your Java programs and instead of generating machine code from your source files it generates byte codes. Byte codes look a lot like machine code but are not specific to any one processor. To execute a Java program you run a program called a byte code

interpreter, which in turn reads the byte code and executes your Java program. The Java byte code interpreter is also called the Java virtual machine or the Java runtime.

Java Byte Codes

Java byte codes are a special set of machine that are not specific to any one processor or computer system instructions produced when you compile a java program.

The advantage of byte codes is that they are not specific to any platform and can therefore be executed on any machine/computer.

The disadvantage of using byte code is in execution speed. Byte codes execute at a slower speed. Because system specific programs run directly on the hardware for which they are compiled, they run significantly faster than Java byte codes which must be processed by the interpreter.

Getting Started Programming in Java

In this section, we are going to write two simple Java programs, a stand alone Java application and an applet that you can view in a Java enabled browser.

Creating An Application In Java

Java applications are general/stand alone programs written in the Java programming language which don't require a browser to run.

NB: Java can be used to write two types of programs: - Java applications and Applet.

Creating the Source File

Java source files are created in a plain text editor e.g. Notepad. These files MUST be saved using a .java file extension.

The following program should be typed in text editor like notepad. The upper and lowercase letters must be used correctly. This is because is case sensitive.

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

For this exercise, we will create a folder called Kenya in C:\ so that we can save our file there. The file name **must** have the same name as the class. (Including the same upper and lower case letters), and must have the extension .java. In this case therefore, our file **must** be called HelloWorld.java. If you call it anything else (even helloworld.java) you will not be able to compile it.

Compiling and Running the Source File

1) Be sure to check which version of Java is installed in your computer (e.g. Jdk1.5.1 or Jdk1.7.1 e.tc.) You'll use this information when setting the path for your Java compiler (Javac)

2) Open MS DOS and type the following at the prompt.

path = C:\Program Files\Java\jdk1.7.0_01\bin

This tells the computer where to find the Java compiler. In this example, I'm assuming that the version of Java installed in your compiler is Jdk1.7.1_01 if it is *jdk1.6.0_25* for example type:-

path = C:\Program Files\Java\jdk1.6.0_25\bin

3) Change to the directory where you saved your source file. In our example we saved it in a folder called Kenya which is in C:\ so our prompt will be C:\Kenya>

Use the Javac command with the name of the file as follows:-

Javac HelloWorld.java

Javac stands for Java compiler and therefore in the above example we are compiling a file called HelloWorld.java

NB: The name must be the same including the same upper and lowercase characters.

If all goes well, you'll end up with a file called *HelloWorld.Class* that's your Java byte code. (You can check this at the same location you saved your source file)

Once you have a class file you can run that file using a Java byte code interpreter. The Java interpreter is simply called Java and you run it from DOS as you did Javac.

4) To run the HelloWorld program, we would do it like this.

java HelloWorld

Note that uppercase and lowercase letters and also note we did **not** include the .class extension.

If your program was typed and compiled correctly, you should get a phrase Hello World!

Creating a Java Applet

Creating Java applets is different from creating a simple Java application. Java applets run and are displayed inside a webpage with other elements, and therefore have special rules for how they behave. As a result, creating an applet may be more complex than creating an application.

Creating the Source File

The source is created in a text editor, compiled to produce the Java byte code file (with a .class extension) which is then placed in a web page (a HTML document).

Creating a Hello World Applet

Create the file below and save it with the same name as the class (including uppercase and lowercase letters.) i.e. HelloWorldApplet. Therefore the name will be HelloWorldApplet.java.

```
import java.awt.Graphics;

public class HelloWorldApplet extends java.applet.Applet
```

```
{
        public void paint(Graphics g)
        {
                g.drawString("Hello World!",5,25);
        }
}
```

Compiling the Source File

The Java applet file is compiled exactly the same way as a Java application.

Change the directory you saved your work and compile using the javac command. In this case it will be: -

javac HelloWorldApplet.java

Including the Applet In a Webpage

If the compilation process was successful, you'll end up with a file called HelloWorldApplet.class in the same directory as your source file. That's your java applet file and it is what you'll include in your webpage.

The HTML With the Applet in it

```
<HTML>
<HEAD>
  <TITLE> Hello to Everyone </TITLE>
</HEAD>

<BODY>
<p> My Java Applet Says:
<APPLET CODE = "HelloWorldApplet.class" WIDTH = 150 HEIGHT = 25>
</APPLET>
</BODY>
</HTML>
```

Note:

CODE – Attribute is used to indicate the name of the class that contains your applet in this case HelloWorldApplet.class

WIDTH and HEIGHT attributes are used to indicate the size of the applet on the page. Here, a box of 150 pixels wide and 25 pixels high is created.

Save the HTML file in the same directory as your class file with a descriptive name (any) and a .html extension e.g. In our case we can call the file HelloWorldApplet.html

Viewing the Result of Running Your Applet

This can be done in two main ways:

1) Open your, web page in a web browser e.g. Mozilla or Internet Explorer. This is the best option because you'll view your applet together with all the text in the web page.

2) Use the **appletviewer** application, which is part of the JDK. The appletviewer is **not** a web browser and will not let you see the entire web page, but its acceptable for testing to see how an applet will look and behave if there is nothing else available.

   To use the appletviewer, change to the directory where your HTML file is located e.g. C:\Kenya then type appletviewer [name of the html file]

   e.g. *c:\Kenya>**appletviewer HelloWorldApplet.html***

Troubleshooting

Here are some of the most common problems you might encounter and how to fix them.

1) 'javac' is not recognized as an internal or external command, operable program or batch file.

   Results when you don't have the JDK's bin directory in your path or the path to that directory is wsrong.

   Solution: Specify the correct path e.g.

   Path = *C:\Program Files\Java\jdk1.7.0_01\bin*

   This depends on the version of Java you are using.

2) Javac : file not found: HelloWorl.java
   Usage: javac<options> <source files>
   Use – help for a list of possible options

   Cause: The filename you are giving to the javac command is not exactly the same name as the file. e.g. you typed javac HelloWorl.java instead of javac HelloWorld.java

3) Error: Could not find or load main class HelloWorld

   This error happens when there is a mismatch between the name of the class defined in the Java file itself and the name of the Java source file.

   In the example above, the file is called HelloWorld.java but the class HelloWorld

   Both cases must match including uppercase and lowercase letters.

4) Other Code Errors

   Go through your program and make sure that the syntax is correct including all uppercase and lowercase letters. Java is case sensitive meaning that uppercase and lowercase letters are treated differently.

Reading Data From the Keyboard

To read data from the keyboard, you may use the InputStream that is referenced by System.in

Example 1

A program that accepts three numbers entered via the keyboard and computes their sum and product.

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

class AddOne
{
  public static void main(String args[])
  {
  try
  {
    BufferedReader s = new BufferedReader(new InputStreamReader(System.in));
    double a, b, c, sum, prod, avg;

    System.out.println("\nEnter three numbers: ");
    a = Double.parseDouble(s.readLine());
    b = Double.parseDouble(s.readLine());
    c = Double.parseDouble(s.readLine());

    sum = a + b + c;
    prod = a * b * c;
    avg = sum / 3;

    System.out.println("\nThe sum of the numbers is " + sum);
    System.out.println("Their product is " + prod);
    System.out.println("Their average is " + avg);
  }
  catch(IOException e){}
  }
}
```

When data is read from the keyboard it is likely that the user may enter data that cannot be handled by the program. As a result, it is advisable to include exceptions in any program that may handle keyboard input. In the above example, the try – catch statement is used to handle the exception.

The program above uses the BufferedReader class of the java.io package to create a buffering character inputStream. In other words, the program creates an object s from the class BufferedReader that stores the values read from the keyboard as implied by the standard InputStreamReader(System.in)

readLine() is the method used to read a line of text which is then converted to a double using the method parseDouble

i.e. *a = Double.parseDouble(s.readLine());*

If it was in integer, we would have used

a = Integer.parseInt (s.readLine());

if it was a float we would have used

a = Float.parseFloat (s.readLine());

If it was a string or a character, there is no need to convert the input to a string or char type.

myname = s.readLine();

if you want to read a character you can use

(char)System.in.read();

Note:

Instead of using:

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

you can simply use

```
import java.io.*;
```

Example 2

A program that accepts the width and length of a rectangle and computes its area and perimeter.

```
import java.io.*;

class MyRectangle
{
  public static void main(String args[])
  {
    try
    {
      BufferedReader s = new BufferedReader(new InputStreamReader(System.in));
      double width,length,area,perim;

      System.out.print("\nEnter the width of the rectangle:> ");
      width = Double.parseDouble(s.readLine());
      System.out.print("Enter the length of the rectangle:> ");
      length = Double.parseDouble(s.readLine());

      area = length * width;
```

```
        perim = 2 * (length + width);

        System.out.println("\nThe area of the rectangle is " + area);
        System.out.println("The perimeter of the rectangle is " + perim);
      }
      catch(IOException e){}
    }
}
```

Example 3

A program that prompts for and outputs someone's age.

```
import java.io.*;

class GetAge
{
  public static void main(String args[])
    try
    {
      BufferedReader s = new BufferedReader(new InputStreamReader(System.in));
      int age;

      System.out.print("\nEnter your age:> ");
      age = Integer.parseInt(s.readLine());

      System.out.println("\nYou are " + age + " years old.\n");
    }
    catch(IOException e){}
  }
}
```

Example 4

A program that prompts for and outputs the grade a student scored.

```
import java.io.*;

class TheGrade
{
  public static void main(String args[])
  {
    try
    {
      char grade;

      System.out.print("\nEnter the grade the student scored:> ");
      grade = (char)System.in.read();
```

```
      System.out.println("\nThe student scored grade " + grade);
    }
    catch(IOException e){}
  }
}
```

Example 5

A program that prompts for and outputs the details of a student.

```
import java.io.*;

class Details
{
  public static void main(String args[])
  {
    try
    {
      BufferedReader s = new BufferedReader(new InputStreamReader(System.in));
      String name, course, place;

      System.out.print("\nEnter your are name: ");
      name = s.readLine();
      System.out.print("Enter the course you are doing: ");
      course = s.readLine();
      System.out.print("Where are you from? ");
      place = s.readLine();

      System.out.println("\nYou are " + name + " from " + place + " and you are doing "
+ course);
    }
    catch(IOException e){}
  }
}
```

Using Scanner to read Input

You can use a Scanner object instead of a BufferedReader object to read values entered by the user. To do so, you have to declare a Scanner object and use one of the Scanner methods to read values.

If you are using a Scanner object, you have to import Scanner using the line shown below:

import java.util.Scanner;

The table below lists some of the common methods in Scanner

Method	Use
nextInt( )	To read number with no decimal point in it.
nextDouble( )	To read number with a decimal point in it (of type double).

Method	Use
nextFloat()	To read number with a decimal point in it (of type float).
nextBoolean()	To read boolean values
nextLong()	To read values of type long
nextShort()	To read values of type short
next( )	To read word (ending in a blank space, for example)
nextLine( )	To read a whole line (or what remains of a line after you've already read some data from the line)
findInLine(".").charAt(0)	To read single character (such as a letter, a digit, or a punctuation character)

Example 1

A program that accepts three numbers entered via the keyboard and computes their sum and product.

```
import java.util.Scanner;

class Add
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double a, b, c, sum, prod, avg;

    System.out.println("\nEnter three numbers: ");
    a = myScanner.nextDouble();
    b = myScanner.nextDouble();
    c = myScanner.nextDouble();

    sum = a + b + c;
    prod = a * b * c;
    avg = sum / 3;

    System.out.println("\nThe sum of the numbers is " + sum);
    System.out.println("Their product is " + prod);
    System.out.println("Their average is " + avg);
  }
}
```

Example 2

A program that accepts the width and length of a rectangle and computes its area and perimeter.

```
import java.util.Scanner;

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

```
  {
    Scanner s = new Scanner(System.in);
    double width,length,area,perim;

    System.out.print("\nEnter the width of the rectangle:> ");
    width = s.nextDouble();
    System.out.print("Enter the length of the rectangle:> ");
    length = s.nextDouble();

    area = length * width;
    perim = 2 * (length + width);

    System.out.println("\nThe area of the rectangle is " + area);
    System.out.println("The perimeter of the rectangle is " + perim);
  }
}
```

Example 3

A program that prompts for and outputs someone's age.

```
import java.util.Scanner;

class Age
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int age;

    System.out.print("\nEnter your age:> ");
    age = myScanner.nextInt();

    System.out.println("\nYou are " + age + " years old.\n");
  }
}
```

Example 4

A program that prompts for and outputs the grade a student scored.

```
import java.util.Scanner;

class GetGrade
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    char grade;
```

```
    System.out.print("\nEnter the grade the student scored:> ");
    grade = myScanner.findInLine(".").charAt(0);

    System.out.println("\nThe student scored grade " + grade);
  }
}
```

Detailed explanation of findInLine(“.”).charAt(0)

Java’s findInLine method looks for things in a line of input. The things the method finds depend on what you put in parentheses. For example, a call to findInLine(“\\d\\d\\d”) looks for a group consisting of three digits. With the following line of code

```
data = myScanner.findInLine(“\\d\\d\\d”);
System.out.println(data);
```

If you type

Testing 123 Testing Testing

The output will be

123

In the call findInLine(“\\d\\d\\d”), each \\d stands for a single digit. This “\\d” is one of many abbreviations in special code called *regular expressions*.

In the world of regular expressions, a dot stands for any character at all. (That is, a dot stands for “any character, not necessarily a dot.”) So findInLine(“.”) tells the computer to find the next character of any kind that the user types on the keyboard.

When you’re trying to input a single character, findInLine(“.”) is very useful.

To grab a single character from the keyboard, one calls findInLine(“.”).charAt(0).

Any findInLine call behaves as if it’s finding a number of characters, not just a single character. Even when you call findInLine(“.”), and the computer fetches just one letter from the keyboard, the Java program treats that letter as one of possibly many input characters.

The call to charAt(0) takes care of the multi-character problem. This charAt(0) call tells

Java to pick the initial character from any of the characters that findInLine fetches.

Example 5

A program that prompts for and outputs the details of a student.

```
import java.util.Scanner;

class StudentDetails
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
```

```
    String name, course, place;

    System.out.print("\nEnter your are name: ");
    name = myScanner.nextLine();
    System.out.print("Enter the course you are doing: ");
    course = myScanner.nextLine();
    System.out.print("Where are you from? ");
    place = myScanner.nextLine();

    System.out.println("\nYou are " + name + " from " + place + " and you are doing " +
course);
  }
}
```

NB:

There is a difference between nextLine( ) and next( )

nextLine( ) reads the entire line include spaces.

next( ) reads only one word. It stops when it gets to a space.

Example:

If you use

name = myScanner.nextLine( );

and enter **John Maina**

both names will be read.

If you use

name = myScanner.next( );

and enter **John Maina**

only the name **John** will be read.

A quick summary of Scanner

The word Scanner is defined in the Java API.

A Scanner is something you can use for getting input.

This Scanner class is new in version 5.0 of the Java API. If you use version Java 1.4.2, then you don't have access to the Scanner class. (You get an error when you try to compile.)

The words System and in are defined in the Java API.

Taken together, the words System.in stand for the computer keyboard.

You can use something like new Scanner(new File("myData.txt")).

System.in is replaced with the words **new File("myData.txt")** because input is from a file as opposed to the keyboard.

The word myScanner doesn't come from the Java API.

The word myScanner is a an object name. Instead of myScanner, you can use readingThing (or any other name you want to use) as long as you use the name consistently. So, if you want to be creative, you can write:

Scanner **readingThing** = new Scanner(System.in);

name = **readingThing.nextLine**( );
System.out.println("Your name is " + name);
Or

Scanner **s** = new Scanner(System.in);

name = **s**.nextLine( );

System.out.println("Your name is " + name);

The line import java.util.Scanner is an example of an *import declaration*.

An optional import declaration allows you to abbreviate names in the rest of your program.

You can remove the import declaration from but if you do, you must use the Scanner class's *fully qualified name* throughout your code. Here's how:

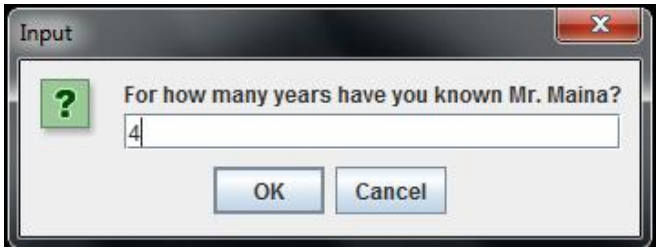
```
class ReadingExample
{
  public static void main(String args[])
  {
    java.util.Scanner myScanner = new java.util.Scanner(System.in);
    String name;

    System.out.print("\nEnter your name: ");
    name = myScanner.nextLine();

    System.out.println("Your name is " + name);
  }
}
```

Getting Input with the JOptionPane Class

You can use the JOptionPane class to display simple dialog boxes such as the one shown below to get text input from the user. Then, you can use the parse methods of the primitive type wrapper classes to convert the text entered by the user to the appropriate primitive type.



Although the **JOptionPane** class has many methods, the only one you need to get simple text input is the **showInputDialog** method. This method uses a single parameter that specifies the prompting message that's displayed in the dialog box. It returns a string value that you can then parse to the proper type.

The JOptionPane class is a part of the **javax.swing** package, so you need to add an import **javax.swing.JOptionPane** statement to the beginning of any program that uses this class.

Example 1

Program that prompts for someone's name and then outputs it.

```
import javax.swing.JOptionPane;

class TheNames
{
  public static void main(String args[])
  {
    String name;

    name = JOptionPane.showInputDialog("Enter your name:");

    System.out.println("\nYour name is " + name);
  }
}
```

Example 2

Program to prompt for and output someone's age.

```
import javax.swing.JOptionPane;

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

    s = JOptionPane.showInputDialog("Enter your age:");
    age = Integer.parseInt(s);

    System.out.println("\nYou are " + age + " years old.");
  }
}
```

Alternative approach

The prompting and conversion of the type can be combined into one line as shown below.

```
import javax.swing.JOptionPane;

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

    age = Integer.parseInt(JOptionPane.showInputDialog("Enter your age:"));

    System.out.println("\nYou are " + age + " years old.");
  }
}
```

Example 3

Program to prompt for the base and height of a triangle and computes and outputs the area.

```
import javax.swing.JOptionPane;

class Triangle
{
  public static void main(String args[])
  {
    double base, height, area;

    base = Double.parseDouble(JOptionPane.showInputDialog("Enter the base of the triangle:"));
    height = Double.parseDouble(JOptionPane.showInputDialog("Enter the base of the triangle:"));

    area = 1.0/2 * base * height;

    System.out.println("\nThe area of the triangle is " + area);
  }
}
```

Java Basics

Java looks a lot like C++ and by extension like C. As a result, much of the syntax is similar to that used in these two languages.

Statements and Expressions

A statement forms a single Java operation

e.g. *int k = 14;*

import Java.awt.Graphics;

System.out.println ("Good Morning Kenya");

All statements in Java and with a semicolon (;)

Variables and Data Types

Variables are locations in memory in which values can be stored. Each variable has a name, type and value.

All variable used in a Java program must be declared.

There are three kinds of variables in Java:

- Instance Variables
- Class variables
- Local variables

i.) Instance Variables – Instance variables are variables that define the attributes of an object.

ii.) Class variables – Class variables are variables that are "owned" by the class and which apply to the class itself and all its instances.

iii.) Local variables – Local variables are variables declared and used inside method definitions

Unlike other languages Java does not have global variables. Instance and class variables can be used to communicate global implementation between and among objects.

Declaring Variables

Variable declaration consists of a type and variable name e.g.

int MyAge;

double x, y, z;

String title;

boolean isAslee;

You can also give an initial value to a variable e.g.

int MyAge = 69;

String MyName = "John";

Variable Data Types

The variable type can be one of the three things: –

i.) One of the eight primitive data types

ii.) The name of a class or interface

iii.) An array

Primitive Data Types

The eight primitive data types handle common types for integers, floating–point numbers, characters and Boolean values (true or false). They are called primitive because they are built into the system and are not actual objects, which makes them more efficient to use.

Integer

Integer (int) is used to hold numbers without a decimal place – whole numbers.

There are four Java *integer types,* each with a different range of values. All are signed, which means they can hold either positive or negative numbers.

Integer Types		
Type	**Size**	**Range**
byte	8 bits	-128 to127
short	16 bits	-32,768 to 32,767
int	32 bits	-2,147,483,648 to 2,147,783,647
long	64 bits	- 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Floating – Point

Floating-point numbers are used for numbers with a decimal part.

There are two floating-point types; float (32 bits, single precision) and double (64 bits, double precision).

Characters

Characters (*char)* are used for individual characters. It is unsigned.

Boolean

The *boolean* type can have one of two values, true or false.

Note:

Primitive types are in lowercase. Be careful when you use them in your programs that you use lowercase because there are also classes with the same names (and initial capital letter) that have different behaviour. For example, the primitive type *boolean* is different from the *Boolean* class.

Class Types

In addition to the eight primitive data types, variables in Java can also be declared to hold an instance of a particular class e.g.

String LastName;

Font BasicFont;

OvalShape anyOval; e.t.c.

Each of these can hold instances of the named class or of any of its subclasses

Comments

Java has three kinds of comments: two for regular comments in source code and one for the special documentation system Javadoc

The symbols /* and */ surround multi-line comments e.g.

```
/* This comment can be
   One or more lines long */
```

Double slashes (//) can be used for a single line of comment .e.g

```
int sum = 0; //The initial sum is zero
```

The final type begins with /** and ends with */. The contents of the special comments are used by the javadoc system but are otherwise identically to the first type of comment.

Literals

A *literal* is a simple value where "what you type is what you get". Numbers, characters and strings are all examples of literals

i.) Number Literals – This include integer literals e.g. 9 and floating – point literal e.g. 6.89

ii.) Boolean literals – Consist of keywords true and false

iii.) Character literals – They are expressed by a single character surrounded by single quotes: 'a', '>', '4' etc.

There are special codes that represent non-printable characters. They are show below.

Character Escape Codes

Escape	Meaning
\n	Newline
\t	Tab
\b	Backspace
\r	Carriage return
\\	Backslash
\'	Single quote
\"	Double quote

String Literals

String literals are a combination of characters in a string. Strings in Java are instances of the class string. They are not arrays of characters as they are in C or C++

An example of a string literal is:

"God bless Kenya"

Expressions and Operators

Expressions – Are statements which return a value when evaluated. They include arithmetic and tests for equality.

Operators – are special symbols are used in expressions to yield new values.

1. Arithmetic Operators

Arithmetic operators are used to perform basic calculations. They are five in number.

Operator	Meaning
+	Addition
-	Substation
*	Multiplication
/	Division
%	Modulus

2. Assignment Operators

Assignment operators include =, ++, – –, +=, –=, *=, /=, %=

There are four forms of assignment: General assignment, multiple assignment, declaration assignment and updation assignment.

General Assignment

A value is assigned to one variable e.g.

```
a = 5;
x = y;
area = width * length;
```

Multiple Assignment

In multiple assignment, one value is assigned to different variables at the same time e.g.

```
x = y = z = 75;
```

The three variables are assigned the value of 75

```
num1 = num2 = num3 = num4 = h + (a * b);
```

The four variables are assigned the result of h + (a * b)'

Declaration Assignment

A variable is declared and assigned an initial value e.g.

```
double sum = 0;
int k = 1;
boolean ans = true;
```

Updation Assignment

The value in a variable is changed and the new value stored in the same variable e.g.

x = x +2; => same as x += 2;

Incrementing and Decrementing

As in C++ and C, the ++ and – operators are used to increment and decrement a variable by 1 respectively e.g. x++ increases the value of x by 1. It is the same as x = x +1

The increment and decrement operators can be prefixed or postfixed, that is ++ or – – can appear before or after the value it increments or decrements.

Examples

x = 20;

y = ++x;

The value of x is increased first and the new value is assigned to y so both x and y will be 21.

x = 20;

y = x++;

The old value of x is assigned to y and then the value of x is increased by 1. Therefore y = 20 and x = 21. The same applies to – –

3. Comparison Operators

Comparison operators are used to compare two values and return a boolean value (i.e. true or false)

Operator	Meaning	Example
= =	Equal	x = = 3
! =	Not Equal	x ! = 3
<	Less than	x < 3
>	Greater than	x > 3
< =	less than or equal to	x < = 3
> =	greater than or equal to	x > = 3

4. Logical Operators

Logical operators are used to combine expressions that result in boolean values. They represent the logical combinations AND, OR, XOR, and Logical NOT

For AND combinations, use either & or && operators. The entire expression will be true only if expressions on either side of the operator are true. If either expression is false, the entire expression is false. When you use & both sides of the expression are evaluated regardless of the outcome.

Using &&, if the left side of the expression is false the entire expression is assumed to be false (the values on the right side don't matter) so the expression returns false and the right side of the expression is never evaluated. This is often called a "short–circuited" expression) it is the one that is commonly used.

For OR expressions, use either | or ||. OR expressions result in true if either or both expressions is true; if both expression operands are false the expression is false. As with & or && the single | evaluates both sides of the expression regardless of the outcome and || is short–circuited – if the left expression is true the expression returns true and the right side is never evaluated.

For XOR (exclusive OR) use the operator ^ which returns true only if operands are different (one true and one false or vice versa) and false otherwise (use if both are true)

For NOT, use the ! operator with a single expression argument. The value of NOT expression is the negative of the expression; if x is true !x is false.

Control Structures

Control structures control the order in which statements are executed in a program.

Control structures include:

1) if…else
2) switch
3) do…while
4) while…
5) for…

If Conditionals

The if conditional statement is used when you want to execute different bits of code based on a certain test. If conditions are nearly identical to if statements in C and C++. They contain the keyword *if* followed by a Boolean test, followed by either a single statement or a block of statements to execute if the test is true.

Syntax 1:

```
if(condition)
  Statement
```

Or

```
if(condition)
{
  Statement 1
  Statement 2
  …
  …
  Statement k
}
```

The condition is tested and if it evaluates to true the statement/block of statement is executed.

Example 1

```
if (x > 0)
  System.out.println(x + " is a positive number.");
```

Example 2

```
if (marks >= 50)
  System.out.println("The student has passed.");
```

Syntax 2:

```
if(condition)
  Statement
else
  Statement
```

Or

```
if(condition)
{
  Statement 1
  Statement 2
  …
  …
  Statement k
}
else
{
  Statement 1
  Statement 2
  …
  …
  Statement h
}
```

The condition is tested and if it evaluates to true the first statement/block of statement is executed and if it evaluates to false the second statement/block of statement is executed.

Example 1

```
if (X >0)
  System.out.println(x + "is a positive number");
else
  System.out.println(x + "Is a negative number");
```

Example 2

```
if (marks >= 50)
  System.out.println("The student has passed.");
else
  System.out.println("The student has failed.");
```

Syntax 3:

```
if(condition_1)
  Statement_1
else if(condition_2)
  Statement_2
else if(condition_3)
  Statement_3
…
…
else if(condition_k)
  Statement_k
else
  Statement
```

Technical Note: In C and C++ the if statement returns an integer (0 or 1) and in Java it returns a Boolean value (True or false)

Example 1

A program that accepts two numbers and outputs the large one.

```
import java.util.Scanner;

class LargeNum
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int a,b,large;

    System.out.print("\nEnter two numbers:> ");
    a = myScanner.nextInt();
    b = myScanner.nextInt();

    if(a > b)
      large = a;
    else
      large = b;

    System.out.println("\nBetween " + a + " and " + b + " the largest is " + large);
  }
}
```

Alternative approach

```
import java.util.Scanner;

class GetLarge
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int a,b;

    System.out.println("\nEnter two numbers:> ");
    a = myScanner.nextInt();
    b = myScanner.nextInt();

    if(a > b)
      System.out.println(a + " is larger than " + b);
    else
      System.out.println(b + " is larger than " + a);
  }
```

```
}
```

Same program using BufferedReader

```
import java.io.*;

class Large
{
  public static void main(String args[])
  {
    try
    {
      BufferedReader s = new BufferedReader(new InputStreamReader(System.in));
      int a,b;

      System.out.println("Enter two numbers:> ");
      a = Integer.parseInt(s.readLine());
      b = Integer.parseInt(s.readLine());

      if(a > b)
        System.out.println(a + " is larger than " + b);
      else
        System.out.println(b + " is larger than " + a);
    }
    catch(IOException e){}
  }
}
```

Example 2

A program that accepts three numbers and outputs the largest.

```
import java.util.Scanner;

class LargestNum
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int a,b,c,largest;

    System.out.println("\nEnter three numbers: ");
    a = myScanner.nextInt();
    b = myScanner.nextInt();
    c = myScanner.nextInt();

    if(a > b && a > c)
      largest = a;
    else if(b > a && b > c)
```

```
      largest = b;
    else
      largest = c;

    System.out.println("\nBetween " + a + ", " + b + " and " + c + " the largest is " +
largest);
   }
}
```

Example 3

A program that accepts marks scored in three subjects, calculates the average and then outputs the grade.

```
import java.util.Scanner;

class StudentsGrades
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double subj1, subj2, subj3, average;
    char grade;

    System.out.println("\nEnter marks in three subjects: ");
    subj1 = myScanner.nextDouble();
    subj2 = myScanner.nextDouble();
    subj3 = myScanner.nextDouble();

    average = (subj1 + subj2 + subj3) / 3.0;

    if(average >= 70)
      grade = 'A';
    else if(average >= 60)
      grade = 'B';
    else if(average >= 50)
      grade = 'C';
    else if(average >= 40)
      grade = 'D';
    else
      grade = 'E';


    System.out.println("\nThe average mark is " + average);
    System.out.println("\nThe grade obtained is " + grade);
  }
}
```

Alternative approach

```
import java.util.Scanner;

class StudentsGrading
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double subj1, subj2, subj3, average;
    char grade;

    System.out.print("\nEnter marks in three subjects: ");
    subj1 = myScanner.nextDouble();
    subj2 = myScanner.nextDouble();
    subj3 = myScanner.nextDouble();

    average = (subj1 + subj2 + subj3) / 3.0;

    if(average >= 70 && average <= 100)
      grade = 'A';
    else if(average >= 60 && average < 70)
      grade = 'B';
    else if(average >= 50 && average < 60)
      grade = 'C';
    else if(average >= 40 && average < 50)
      grade = 'D';
    else
      grade = 'E';


    System.out.println("\nThe average mark is " + average);
    System.out.println("\nThe grade obtained is " + grade);
  }
}
```

Example 4

A program to illustrate the use of "And"

```
import java.util.Scanner;

class AndExample
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double marks;
```

```
    System.out.print("\nEnter marks the student scored: ");
    marks = myScanner.nextDouble();

    if(marks >= 0 && marks <= 100)
      System.out.println(marks + " is a VALID mark.\n");
    else
      System.out.println(marks + " is an INVALID mark.\n");
  }
}
```

Example 5

A program to illustrate the use of "Or"

```
import java.util.Scanner;

class OrExample
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double marks;

    System.out.print("\nEnter marks the student scored: ");
    marks = myScanner.nextDouble();

    if(marks < 0 || marks > 100)
      System.out.println(marks + " is a INVALID mark.\n");
    else
      System.out.println(marks + " is an VALID mark.\n");
  }
}
```

Example 6

A program to illustrate the use of "Exclusive Or"

```
import java.util.Scanner;

class ExclusiveOr
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int a,b;

    System.out.print("\nEnter two integers: ");
    a = myScanner.nextInt();
    b = myScanner.nextInt();
```

```
    if(a == 0 ^ b == 0)
      System.out.println("\nOnly one of the values is equal to zero.");
    else
      System.out.println("\nEither both values are zero or non-zero.");
  }
}
```

The Conditional Operator/Ternary Operator

The alternative to using the if and else keywords in conditional statements is to use the conditional operator, sometimes called the ternary operator (because it has three parts). It is most useful for very short or simple conditional and looks like this: -

test?trueresult:falseresult;

Test – Test is a boolean expression that returns true or false, just like the test in the if statement.

The value of *trueresult* is returned if the test is true and if it is false the value of *falseresult* is returned if the test is false.

Example

The following conditional tests the values of x and y returns the smaller of the two and assigns the value to the variable small.

small = x < y?x:y;

Example 1

A program that accepts two integers and outputs the largest.

```
import java.util.Scanner;

class LargeCon
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double num1, num2, large;

    System.out.println("\nEnter two numbers: ");
    num1 = myScanner.nextDouble();
    num2 = myScanner.nextDouble();

    large = num1 > num2?num1:num2;

    System.out.println("\nBetween " + num1 + " and " + num2 + " the largest is " +
large);
  }
}
```

Switch

Switch transfers control in multiple directions. It is used in place of nested if statements but if has some restrictions.

1) It only works with values that evaluates to int (byte, char, short or int). It does not work with long, float, Strings)
2) It is only used to test for equality.

Syntax

```
Switch (expression)
{
        Case Constant1:
                Block of statements;
                break;
        Case Constant2:
                Block of statements;
                break
        .....
        ....
        default:
                Block of statements;
}
```

Comparing If and Switch

The code below is first written using nested if…else statements and then re-written using switch.

```
if (oper = = '+')
        ans = a + b;
else if (oper = = '–')
        ans = a – b;
else if (oper = = '*')
        ans = a * b;
else if (oper = = '/')
        ans = a / b;
else
        System.out.println("Invalid Operator");
```

Same Code Using Switch

```
Switch (oper)
{
case '+' ans = a +  b;
        break;
case '–' ans = a – b;
        break;
case '*' ans = a * b;
```

```
        break;
case '/' ans = a / b;
        break;
default: System.out.println("Invalid operator");
```

Example 1

A program that accepts the points scored and then determines the award using the system below

Points	Award
4	Distinction
3	Credit
2	Pass
1	Fail

```
import java.util.Scanner;

class AwardSwitch
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int points;

    System.out.print("\nEnter the points the student scored:> ");
    points = myScanner.nextInt();

    switch(points)
    {
    case 4: System.out.println("\nThe student got a Distinction.");
          break;
    case 3: System.out.println("\nThe student got a Credit.");
          break;
    case 2: System.out.println("\nThe student got a Pass.");
          break;
    case 1: System.out.println("\nThe student got a Fail.");
          break;
    default: System.out.println("\nThe points you entered are invalid.");
          break;
    }
  }
}
```

Example 2

A program that accepts the points scored by a student and then determines if the student has passed (4, 3 or 2 points) of failed (1 point).

```
import java.util.Scanner;

class AwardSwitchTwo
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int points;

    System.out.print("\nEnter the points the student scored:> ");
    points = myScanner.nextInt();

    switch(points)
    {
    case 4:
    case 3:
    case 2: System.out.println("\nThe student Passed.");
          break;
    case 1: System.out.println("\nThe student Failed.");
          break;
    default: System.out.println("\nThe points you entered are invalid.");
          break;
    }
  }
}
```

Example 3

A program that accepts a character and checks whether it s a vowel.

```
import java.util.Scanner;

class VowelProg
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    char lett;

    System.out.print("\nEnter a character: ");
    lett = myScanner.findInLine(".").charAt(0);

    switch(lett)
    {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
```

```
    case 'u':System.out.println("\n" + lett + " is a vowel.");
          break;
    default: System.out.println("\n" + lett + " is not a vowel.");
          break;
    }
  }
}
```

Approach 2

Same program without using a Scanner object.

```
import java.io.*;

class Vowel
{
  public static void main(String args[])
  {
    try
    {
      char lett;
      System.out.println("Enter a character:>");
      lett = (char)System.in.read();
      switch(lett)
      {
      case 'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':System.out.println(lett + " is a vowel.");
            break;
      default: System.out.println(lett + " is not a vowel.");
            break;
      }
    }
    catch(IOException e){}
  }
}
```

The programs above will only recognize lowercase vowels. When a vowel is entered in uppercase, it is flagged as a non-vowel.

To convert a vowel to lower case, you use the toLowerCase() function as shown below;

new_lett = Character.toLowerCase(lett);

In the above example, new_lett is assigned the lowercase version of lett.

Example 4

The program below is a rewritten version of the vowel program. The program works with both uppercase and lowercase characters.

```
import java.util.Scanner;

class VowelProgTwo
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    char lett;

    System.out.print("\nEnter a character: ");
    lett = myScanner.findInLine(".").charAt(0);

    switch(Character.toLowerCase(lett))
    {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':System.out.println("\n" + lett + " is a vowel.");
          break;
    default: System.out.println("\n" + lett + " is not a vowel.");
          break;
    }
  }
}
```

Example 5

A program that accepts the grade a student scored and assigns an award using the system below.

Grade	Award
A	Distinction
B	Credit
C	
D	Pass
E	Fail

```
import java.util.Scanner;

class SwitchGrade
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    char grade;

    System.out.print("\nEnter the grade the student got:> ");
```

```
    grade = myScanner.findInLine(".").charAt(0);

    switch(grade)
    {
    case 'A': System.out.println("\nThe student got a Distinction.");
          break;
    case 'B':
    case 'C': System.out.println("\nThe student got a Credit.");
          break;
    case 'D': System.out.println("\nThe student got a Pass.");
          break;
    case 'E': System.out.println("\nThe student got a Fail.");
          break;
    default: System.out.println("\nThe grade you entered are invalid.");
           break;
    }
   }
}
```

You can also **Character.toUpperCase(grade)** to ensure that the program works with grades entered in both uppercase and lowercase.

Loops

Loops are used to execute a statement or a block of statements repeatedly as long as a given condition is true.

There are three types of loop structure in Java. These are;

i.) do…while

ii.) while…

iii.) for…

A loop has three main components;

i.) Initialization – This specifies where the loop starts.

ii.) Updation – This specifies how the value controlling the loop changes.

iii.) Condition – This specifies the condition that cause the loop to stop.

do…while loop

Syntax

Initialization

```
do
{
        Statement 1
        Statement 2
        Statement 3
        ….
```

```
        ….
        Statement k
        Updation
}while(condition);
```

Example 1

A program that prints numbers between 1 and 20.

```
class DoOne
{
  public static void main(String args[])
  {
    int k;

    k = 1;//Initialization

    do
    {
      System.out.println("k = " + k);
         k = k + 1;//updation
    }while(k <= 20); //Condition
  }
}
```

Example 2

A program that prints 5, 10, 15,….300

```
class DoTwo
{
  public static void main(String args[])
  {
    int k;

    k = 5;

    do
    {
      System.out.println("k = " + k);
         k = k + 5;
    }while(k <= 300);
  }
}
```

Example 3

A program that prints 5, 10, 20, 40, 80, 160, 320, 640

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

```
  {
    int k;

    k = 5;

    do
    {
      System.out.println("k = " + k);
         k = k * 2;
    }while(k <= 640);
  }
}
```

Example 4

A program that prints 600, 300, 150,…

```
class DoFour
{
  public static void main(String args[])
  {
    int k;

    k = 600;

    do
    {
      System.out.println("k = " + k);
         k = k / 2;
    }while(k >= 10);
  }
}
```

Example 5

A program that computes the sum of all numbers between 1 and 100.

```
class DoSum100
{
  public static void main(String args[])
  {
    int k, sum = 0;

    k = 1;

    do
    {
      sum = sum + k;
         k = k + 1;
```

```
    }while(k <= 100);

    System.out.println("\nThe sum of all numbers between 1 and 100 is " + sum);
  }
}
```

Example 6

A program that computes the sum of all even numbers between 1 and 100.

```
class DoSumEven100
{
  public static void main(String args[])
  {
    int h, sum = 0;

    h = 1;

    do
    {
      if(h % 2 == 0)
        sum = sum + h;
        h = h + 1;
    }while(h <= 100);

    System.out.println("\nThe sum of all numbers between 1 and 100 is " + sum);
  }
}
```

Example 7

A program that prints all number divisible by 5 between 100 and 200.

```
class PrintDivBy5
{
  public static void main(String args[])
  {
    int p;

    System.out.println("\nThe numbers divisible by 5 between 100 and 200 are:\n");

    p = 100;

    do
    {
      if(p % 5 == 0)
        System.out.print(p + " ");
        p = p + 1;
    }while(p <= 200);
  }
```

```
}
```

while… loop

Syntax

Initialization

```
while(condition)
{
        Statement 1
        Statement 2
        Statement 3
        ….
        ….
        Statement k
        Updation
}
```

Example 1

A program that prints all numbers between 1 and 20.

```
class WhileOne
{
  public static void main(String args[])
  {
    int k;

    k = 1;//Inititalization

    while(k <= 20) //Condition
    {
      System.out.println("k = " + k);
         k = k + 1;//Updation
    }
  }
}
```

Example 2

A program that prints 5, 10, 15, 20,…300

```
class WhileTwo
{
  public static void main(String args[])
  {
    int k;

    k = 5;

    while(k <= 300)
```

```
    {
      System.out.println("k = " + k);
         k = k + 5;
    }
  }
}
```

Example 3

A program to print all numbers divisible by 5 between 100 and 200

```
class WhilePrintDivBy5
{
  public static void main(String args[])
  {
    int p;

    System.out.println("\nThe numbers divisible by 5 between 100 and 200 are:\n");

    p = 100;

    while(p <= 200)
    {
      if(p % 5 == 0)
        System.out.print(p + " ");
         p = p + 1;
    }
  }
}
```

Example 4

A program that computes the sum of all numbers between 1 and 100

```
class WhileSum100
{
  public static void main(String args[])
  {
    int k,sum = 0;

    k = 1;

    while(k <= 100)
    {
      sum = sum + k;
         k = k + 1;
    }

    System.out.println("\nThe sum of all numbers between 1 and 100 is " + sum);
```

```
  }
}
```

Difference between do…while and while… loop structures

In do…while loop, the condition is tested at the end of the loop and as a result, do…while loop will always be executed at least once even if the condition evaluates to false.

In while… loop the condition is tested at the beginning of the loop and therefore the loop is only executed if the condition evaluates to true.

Example 1

```
class DiffDo
{
  public static void main(String args[])
  {
    int k;

    k = 100;

    do
    {
      System.out.println("\nk = " + k);
         k = k + 1;
    }while(k <= 20);

    System.out.println("\nGoodbye.\n");
  }
}
```

Output

```
K = 100
Goodbye.
```

Example 2

Same program but this time using the while… loop.

```
class DiffWhile
{
  public static void main(String args[])
  {
    int k;

    k = 100;

    while(k <= 20)
    {
      System.out.println("\nk = " + k);
      k = k + 1;
```

```
    }

    System.out.println("\nGoodbye.\n");
  }
}
```

Output

Goodbye.

for…loop

Syntax

```
for(initialize; condition; updation)
{
        Statements;
}
```

Example 1

A program to print "Hello World" 10 times.

```
class ForOne
{
  public static void main(String args[])
  {
    int i;

    for(i = 1;i <= 10;i++)
      System.out.println("Hello World");
  }
}
```

Example 2

A program that prints all numbers between 1 and 10.

```
class ForTwo
{
  public static void main(String args[])
  {
    int i;

    for(i = 1;i <= 10;i++)
      System.out.println("i = " + i);
  }
}
```

Example 3

A program that prints 10, 9, 8,… 1

```
class ForThree
{
  public static void main(String args[])
  {
    int i;

    for(i = 10;i >= 1;i– –)
      System.out.println("i = " + i);
  }
}
```

Example 4

A program that prints 100, 95, 90,… 10

```
class ForFour
{
  public static void main(String args[])
  {
    int i;

    for(i = 100;i >= 10;i = i – 5)
      System.out.println("i = " + i);
  }
}
```

Example 5

A program that 10000, 2000, 400,…

```
class ForFive
{
  public static void main(String args[])
  {
    int h;

    for(h = 10000;h >= 5;h = h / 5)
      System.out.println("h = " + h);
  }
}
```

Example 6

Program to print all numbers divisible by 6 between 400 and 600

```
class ForDiv6
{
  public static void main(String args[])
  {
    int h;
```

```
    System.out.println("\nThe numbers divisible by 6 between 400 and 600 are:\n");

    for(h = 100;h <= 400;h++)
      if(h % 6 = = 0)
        System.out.print(h + " ");
  }
}
```

Example 6

A program that computes the sum of all numbers between 1 and 100

```
class ForSum100
{
  public static void main(String args[])
  {
    int h;

    int i, sum = 0;
    for(i = 1;i <= 100;i++)
      sum = sum + i;

    System.out.println("\nThe sum of all numbers between 1 and 100 is " + sum);
  }
}
```

Example 7

A program computes the sum of all even numbers between 1 and 100

```
class ForSumEven100
{
  public static void main(String args[])
  {
    int i, sum = 0;

    for(i = 1;i <= 100;i++)
      if(i % 2 = = 0)
        sum = sum + i;

    System.out.println("\nThe sum of all even numbers between 1 and 100 is " + sum);
  }
}
```

Example 8

A program computes the factorial of a number (n! = 1 * 2 * 3 *…* n)

```
import java.util.Scanner;

class GetFactorial
```

```
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int i,num, fact = 1;

    System.out.print("\nEnter an integr to find factorial: ");
    num = myScanner.nextInt();

    for(i= 1;i <= num;i++)
      fact = fact * i;
    System.out.println("\nThe factorial of " + num + " is " + fact);
  }
}
```

Break and Continue statements

Break causes an immediate exit from the innermost loop. When break is encountered, the program immediately exits from the innermost loop even if the conditional test for the loop has not evaluated to false.

Continue causes the loop to skip to the next iteration. When continue is encountered, the program ignores the remaining statements for the current iteration and immediately skips to the next iteration.

Example 1

A program illustrating the use of the break statement.

```
class ForBreak
{
  public static void main(String args[])
  {
    int i;

    for(i = 1;i <= 10;i++)
    {
      if(i = = 6)
          {
            System.out.println("\nThis is my favourite number.\n");
            break;
          }
          System.out.println("i = " + i);
    }
  }
}
```

Output

```
i = 1
i = 2
```

```
i = 3
i = 4
i = 5
This is my favourite number.
```

Example 2

A program illustrating the use of the continue statement.

```
class ForContinue
{
  public static void main(String args[])
  {
    int i;

    for(i = 1;i <= 10;i++)
    {
      if(i = = 6)
          {
            System.out.println("\nThis is my favourite number.\n");
            continue;
          }
          System.out.println("i = " + i);
    }
  }
}
```

Output

```
i = 1
i = 2
i = 3
i = 4
i = 5

This is my favourite number.

i = 7
i = 8
i = 9
i = 10
```

Nested Loops

You can have a loop inside another loop. This are called nested loops. In some instances, e.g. when working with matrices, a programmer is forced to use nested loops.

Example 1

A program that prints the mathematical table of 1 to 10.

```
class MathematicalTable
{
```

```
  public static void main(String args[])
  {
    int i,j;

    for(i = 1;i <= 10;i++)
    {
      for(j = 1;j <= 15;j++)
            System.out.println(i + " * " + j + " = " + (i * j));
      System.out.println("\n");
    }
  }
}
```

Arrays

An array is a data structure that holds a number of elements of the same data type under a common name.

Unlike in other languages, arrays in Java are actual objects that can be passed round and treated like other objects.

To create arrays in Java you use three steps: -

i.) Declare a variable to hold the array

ii.) Create a new array object and assign it to the array variable.

iii.) Store data in the array.

i.) Declaring Array Variables

The first step in creating an array is creating a variable that will hold the array, just as you would any other variable.

Array variable indicate the type of object that array will hold. (Just as they do for any variable) and the name of the array followed by the empty square brackets ([ ]).

Examples

```
String stud_name[ ];

int marks[ ];

double ages [];
```

An alternative way of defining an array variable is to put the square bracket after the type instead of after the variable. They are equivalent but this later form is much more readable.

Examples (Rewritten)

```
String[ ] stud_name;

int[ ] marks;

double[ ] ages;
```

ii.) Creating Array Objects

The second step is to create an array object and assign it to that variable. This can be done in two main ways.

a) Using new

b) Directly initializing the contents of that array.

a) Using The New Operator

Example 1

```
stud_name = new String[20];

//This line assumes you have already declared Stud_name as shown above.
```

The line above creates a new array of strings with 20 slots (elements). When you create a new array object using new you MUST indicate how many slots/elements that array will hold.

Example 2

marks = new int[30];

//This line assumes you have already declared marks as shown above.

The statement above declares an array called marks that can hold a maximum of 30 integers.

When you create an array object using *new* all its slots are initialized for you (0 for numeric arrays, false for Boolean, '\0' for character arrays and null for objects).

You can then assign actual values or objects to the slots in that array.

b) Directly initializing the contents of that array.

You can create an array and initialize its contents at the same time. Instead of using new to create a new array object, enclose the elements of the array inside braces separated by commas e.g.

String[ ] names = {"John", "Maina", "Alex", "Alice", "Jane"};

An array the size of the number of elements you have included will be automatically, created for you. In the above example, the array size will be set to 5.

double[ ] wages = {250, 450, 800, 670, 900, 650};

Declaring and Creating an Array object using one statement.

Using the approach explained above, you need two lines/statements to declare and create an array as show below.

Example 1

String[ ] stud_name;
stud_name = new String[20];

Example 2

int[ ] marks;
marks = new int[30];

The two statements can be combined into one if need be. See examples below:

Example 1

String[ ] stud_name = new String[20];

Example 2

int[ ] marks = new int[30];

Accessing Array Elements

To access an array element, you use an array subscript in square brackets. This subscript starts from 0 and ends at array size – 1. In the array met earlier (names – see above), to

access the first element (John) we use names[0]. To access the fifth element (Jane) we use names[4].

If you try to access an element that is outside the boundary e.g. names[6] in the above program, you'll get an error. Java helps one to avoid making such errors by providing the length instance variable which returns the length of any array object.

Example

int k;

k = names.length;

The statement above returns the size of the array called names (which is 5 in our case).

Example 1

```
class ArrayOne
{
  public static void main(String args[])
  {
    String[] jina = new String[4];
    int i;

    jina[0] = "John";
    jina[1] = "Alex";
    jina[2] = "Alice";
    jina[3] = "Atieno";

    System.out.println("\nThe names in the array are:-");
    for(i = 0;i < 4;i++)
      System.out.println(jina[i]);
  }
}
```

Example 2

```
class ArrayNumOne
{
  public static void main(String args[])
  {
    int[ ] numbers = new int[5];
    int i;

    numbers[0] = 10;
    numbers[1] = 25;
    numbers[2] = 32;
    numbers[3] = 15;
    numbers[4] = 95;

    System.out.println("\nThe values in the array are:-\n");
    for(i = 0;i < 5;i++)
```

```
      System.out.println(numbers[i]);
  }
}
```

Example 3

In this example, the array is initialized at the time of declaration (and hence no need to use the new operator)

```
class ArrayNumTwo
{
  public static void main(String args[])
  {
    int[ ] numbers = {23,45,56,47,55};
    int i;

    System.out.println("\nThe values in the array are:-\n");
    for(i = 0;i < 5;i++)
      System.out.println(numbers[i]);
  }
}
```

Example 4

A program that accepts 10 values entered via the keyboard, stores them in an array and then outputs them.

```
import java.util.Scanner;

class ArrayNumbers
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int[ ] numbers = new int[10];
    int i;

    for(i = 0;i < 10;i++)
    {
      System.out.print("Enter element " + (i + 1) + ":> ");
      numbers[i] = myScanner.nextInt();
    }

    System.out.println("\nThe values you entered were:-\n");
    for(i = 0;i < 10;i++)
      System.out.print(numbers[i] + " ");
  }
}
```

Example 5

A program that accepts marks for 10 students stores them in an array and then outputs them along with the total and average marks.

```
import java.util.Scanner;

class ArrayMarks
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double[ ] marks = new double[10];
    double sum = 0, avg;
    int i;

    for(i = 0;i < 10;i++)
    {
      System.out.print("Enter the marks for student " + (i + 1) + ":> ");
      marks[i] = myScanner.nextDouble();
      sum = sum + marks[i];
    }

    avg =  sum / 10;

    System.out.println("\nAnalysis of the performance:-\n");
    for(i = 0;i < 10;i++)
      System.out.println("Student " + (i + 1) + ": " +marks[i] + " ");

    System.out.println("\n\nThe total mark is " + sum);
    System.out.println("The average mark is " + avg);
  }
}
```

Example 6

A program that prompts the user for the number of elements he wants to work with, prompts him for the values then outputs them along with their sum, the average, the largest and the smallest value.

```
import java.util.Scanner;

class ArraySum
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    double[] numbers;
    double sum = 0, avg, largest, smallest;
```

```
    int i, all;

    System.out.print("How many numbers do you want to work with? ");
    all = myScanner.nextInt();

    numbers = new double[all];

    for(i = 0;i < all;i++)
    {
      System.out.print("Enter element " + (i + 1) + ":> ");
      numbers[i] = myScanner.nextDouble();
    }

    largest = smallest = numbers[0];

    for(i = 0;i < all;i++)
    {
      sum = sum + numbers[i];
      if(numbers[i] > largest)
        largest = numbers[i];
      else if(numbers[i] < smallest)
        smallest = numbers[i];
    }

    avg =  sum / all;

    System.out.println("\nThe values you entered are:-\n");
    for(i = 0;i < all;i++)
      System.out.print(numbers[i] + " ");

    System.out.println("\n\nTheir sum is " + sum);
    System.out.println("Their average is " + avg);
    System.out.println("The largest number is " + largest);
    System.out.println("The smallest number is " + smallest);
  }
}
```

Multidimensional Arrays

Multidimensional arrays in Java are treated as an array of arrays e.g.

int arr[ ][ ] = new int[10] [10];

This is an example with a two dimensional arrays with a maximum of 10 rows and 10 columns.

Example 1

A program that accepts values into a 3 × 4 matrix and then outputs them.

```
import java.util.Scanner;

class Matrix
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int[ ][ ] mtx = new int[3][4];
    int i,j;

    for(i = 0;i < 3;i++)
    {
      for(j = 0;j < 4;j++)
      {
        System.out.print("Enter element (" + (i + 1) + "," + (j + 1) + ") :> ");
        mtx[i][j] = myScanner.nextInt();
      }
    }

    System.out.println("\nThe matrix is:\n");

    for(i = 0;i < 3;i++)
    {
      for(j = 0;j < 4;j++)
      {
        System.out.print(mtx[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

Initializing a two dimensional array

A two dimensional array is initialized row by row as shown below

int[ ][ ] arr = {{10,20},{30,15},{25,30}};

In the above example, a 3 × 2 [3 row, 2 columns] is declared and initialized with values.

Example 1

In the program below a 3 × 4 array is initialized with values and then the values along with their sum and average is output.

```
import java.util.Scanner;

class MatrixInitialize
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);
    int[][] data = {{8,7,4,5},{6,3,8,9},{8,2,6,9}};
    int i,j,sum = 0;
    double avg;

    for(i = 0;i < 3;i++)
    {
      for(j = 0;j < 4;j++)
      {
        sum = sum + data[i][j];
      }
    }

    avg = sum / 12.0;

    System.out.println("\nThe matrix is:\n");

    for(i = 0;i < 3;i++)
    {
      for(j = 0;j < 4;j++)
      {
        System.out.print(data[i][j] + " ");
      }
      System.out.println();
    }

    System.out.println("\nThe sum of the elements is " + sum);
    System.out.println("Their average is " + avg);
  }
}
```

Object Oriented Programming in JAVA

Definition of Object-Oriented Programming

The term *object-oriented programming* refers to a type of computer programming based on the premise that all programs are essentially computer-based simulations of real-world objects or abstract concepts.

Understanding Objects

Objects – both in the real world and in the world of programming – are entities that have certain basic characteristics. An object has identity, type, state, and behavior.

Objects have identity

Every object in an object-oriented program has an *identity*. In other words, every occurrence of a particular type of object – called an *instance* – can be distinguished from every other occurrence of the same type of object, as well as from objects of other types.

In the real world, object identity is a pretty intuitive and obvious concept. If you have two cars, you know that although both of them are cars (that's their object type), they are not the same car. Each has a distinct identity. They can both be of the same make, the same colour etc. but they are not the same car! They might have small variations; One might have a torn seat while the other one does not. One may have new tyres and the other one does not.

In object-oriented programming, each object has its own location in the computer's memory. Thus, two objects, even though they may be of the same type, have their own memory location. The address of the starting location for an object provides us with a way of distinguishing one object from another, because no two objects can occupy the same location in memory.

Objects have type

Object-oriented programming lets you assign names to the different kind of objects in a program. In Java, types are defined by classes. So when you create an object from a type, you're saying that the object is of the type specified by the class. For example, the following statement creates an object of type Invoice:

```
Invoice my_invoice = new Invoice( );
```

Then, the identity of this object (that is, its address in memory) is assigned to the variable my_invoice, which the compiler knows can hold references to objects of type Invoice.

Objects have state

The type of an object determines what attributes the object has. Thus, all objects of a particular type have the same attributes. However, they don't necessarily have the same values for those attributes.

For example, all objects of type Student have an admission number, name, date of birth, religion etc. but two Student objects have different values for these attributes.

The combination of the values for all the attributes of an object is called the object's ***state***. Unlike its identity, an object's state can and usually does change over its lifetime.

For example, a student can change his/her name. The address and phone number of an employee can change etc.

Objects have behavior

Objects have *behavior,* which means they can do things. Like state, the specific behavior of an object depends on its type. But unlike state, the behavior is the same for each instance of a type.

For example, if two people have similar calculators, they can use the calculators to add any two numbers of their choosing. Although the numbers and results might be different the mechanism of adding (behavior) is the same. In other words, they all have a different state, but the same behavior.

Definitions:

Object – An object a variable with data and methods associated with it. It can also be defined as an instance of a class.

Class – A class is a template for an object, which contains variables and methods representing behaviour and attributes.

Instance – An instance is a concrete representation of a class.

NB: Instance and objects are the same thing.

Class Library – A class library is a collection of classes that come with Java and which are intended to be used repeatedly in different programs.

Behaviour and Attributes – Every class you write in Java has two basic features: - attributes and behaviour.

Attributes – Attributes are individual things that differentiate one object from another and determine the appearance, state or other qualities of that object. Examples include age, color, height etc.

Attributes are defined in classes by variables. Those variables types are defined in the class and each object can have its own values for those variables. Because each instance of a class can have different values for its variables, its variables are often called instance variables.

i.) Instance Variables – Instance variables are variables that define the attributes of the object. Their types and name are defined in the class, but their values are set and changed in the object.

ii.) Class Variables – Class variables are variables "owned" by the class and which apply to the class itself and all its instances.

Behaviour – A class's behaviour determines how an instance of the class operates; for example how it will "react" if asked to do something or if its internal state changes. Behaviour is the only way objects can do anything to themselves or have anything done to them. Behaviour could include things like start, stop, speed up etc.

To define an object's behaviour, you create methods, a set of Java statements that accomplish some task.

Methods look and behave just like functions in other languages but are defined and are accessible solely inside a class. Java does not have functions defined outside classes (as C++ does)

Methods – Methods are functions defined inside a class and that operate on instances of the classes.

Instance Methods – Instance methods are methods defined in a class, which operates on an instance of that class. They are the ones that are commonly used and are usually just called methods.

Class Methods – Class methods are methods defined in a class, which operates on the class itself and can be called via the class or any of its instances.

Interface – The interface of a class is the set of methods and fields that the class makes public so other objects can access them.

Implementation – Implementation refers to exactly how an object does what it does. It consists of the private members of the class and definition [code] of all the methods [both public and private]. Implementation can and should be hidden within the object. Someone who uses the object needs to know **what** the object does, but doesn't need to know **how** it works. If you later find a better way for the object to do its job, you can swap in the new improved version without anyone knowing the difference.

Creating a class

All Java programs are classes.

Declaring a class

All classes must be defined by a *class declaration* that provides the name for the class and the body of the class.

Basic form of a class

Declaration:

[public] class *ClassName*

{

class-body

}

The public keyword indicates that this class is available for use by other classes. The public keyword is optional.

Picking class names

The *ClassName* is an identifier that provides a name for your class. You can use any identifier you want to name a class, but the following three guidelines can simplify the process:

- **Begin the class name with a capital letter.** If the class name consists of more than one word, capitalize each word e.g. Ball, BankAccount and PartTimeStudent.

- **Whenever possible, use nouns for your class names.** Classes create objects, and nouns are the words you use to identify objects. Thus, most class names should be nouns.
- **Avoid using the name of a Java API class.** There is no rule that prohibits creating a class with the same name as a Java API class, but if you do, you have to use fully qualified names (like java.util.Scanner) to tell your class and the API class with the same name apart. There are literally thousands of Java API classes, so avoiding them all is very hard, but at the least, you should avoid commonly used Java class names (e.g. String, Math etc.) as well as any API classes your application is likely to use.

Example

To create a class you simply write the class key word followed by the name of the class. Inside the class you include the instance variables and methods.

```
class Employee
{


}
```

In the above example, the class does nothing because it is empty. We are now going to include instance variables to determine/show the attributes of an Employee e.g. name, salary etc. and methods to represent the behaviour and state of the Employee.

Below is an example of the Employee class with instance variables and methods included.

Example 1

```
import java.util.Scanner;

class Employee
{
  private String emp_id, emp_name;
  private int year_of_birth;
  private double basic_salary, allowances, gross, tax_amount, net_salary;

  public void input()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the employee's employee ID: ");
    emp_id = myScanner.nextLine();
    System.out.print("Enter his/her name: ");
    emp_name = myScanner.nextLine();
    System.out.print("Enter his/her year of birth: ");
    year_of_birth = myScanner.nextInt();
    System.out.print("Enter his/her basic salary: ");
```

```
    basic_salary = myScanner.nextDouble();
    System.out.print("Enter his/her allowances: ");
    allowances = myScanner.nextDouble();
  }

  public void compute()
  {
    gross = basic_salary + allowances;

    if(gross >= 20000)
      tax_amount = 10.0 / 100 * gross;
    else
      tax_amount = 0;

    net_salary = gross - tax_amount;
  }

  public void output()
  {
    System.out.println("Employee ID: " + emp_id);
    System.out.println("Name: " + emp_name);
    System.out.println("Year of Birth: " + year_of_birth);
    System.out.println("Basic Salary: " + basic_salary);
    System.out.println("Allowances: " + allowances);
    System.out.println("Gross Salary: " + gross);
    System.out.println("Tax Amount: " + tax_amount);
    System.out.println("Net Salary: " + net_salary);
  }
}
```

The above class is complete and will compile properly but if you try to run it you will get an error message. This is because, when you run a compiled Java class directly Java assumes it is an application and looks for a main() method. That is where execution begins in an application.

To do something with the Employee class, we need to create an instance of that class, create a separate Java applet or application or simply add a main() method to the class. The latter [adding main()] is the simplest (but not necessarily the best).

The example below shows a separate application that makes use of the Employee class.

```
class MyCompany
{
  public static void main(String args[])
  {
    Employee emp1 = new Employee();

    System.out.println("\nData entry for the employee:");
    System.out.println("-----------------------------------------");
```

```
    emp1.input();
    emp1.compute();

    System.out.println("\nData analysis for the employee:");
    System.out.println("-----------------------------------------");
    emp1.output();
  }
}
```

The two files [Employee and MyCompany] should be saved in the same folder.

NB: Employee emp1 = new Employee() creates a new instance of the Employee class and stores a reference to it in the variable emp1. Remember, you don't usually operate directly on the classes in your Java program; instead you create objects from those classes and then call methods on those objects.

Approach 2

Instead of having two different files [e.g. Employee and MyCompany in the example above], you can have the class and the main() method in the same file.

Although this approach is simpler, it has one constraint/challenges. The main method is treated as a member function [method] and therefore even private members are directly accessible in the main method.

Example 2

```
import java.util.Scanner;

class Student
{
  private String adm_no, name;
  private int no_of_units;
  private double fee_paid;

  public void input()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the student's admission number: ");
    adm_no = myScanner.nextLine();
    System.out.print("Enter his/her name: ");
    name = myScanner.nextLine();
    System.out.print("Enter the number of units he/she is doing: ");
    no_of_units = myScanner.nextInt();
    System.out.print("Enter the fee he/she has paid: ");
    fee_paid = myScanner.nextDouble();
  }

  public void output()
```

```
  {
    System.out.println("Admission Number: " + adm_no);
    System.out.println("Name: " + name);
    System.out.println("No of Units: " + no_of_units);
    System.out.println("Fee Paid: " + fee_paid);
  }

  public static void main(String args[])
  {
    Student stud1 = new Student();

    System.out.println("\nData entry for the student:");
    System.out.println("-----------------------------------------");
    stud1.input();

    System.out.println("\nData analysis for the student:");
    System.out.println("-----------------------------------------");
    stud1.output();
  }
}
```

Approach 3

To remedy the problem above, you can have two different classes – one that defines the objects and one that contains the main() method but save them in the same file. The file should be saved using the name of **the class that contains the main method**.

Example 3

```
import java.util.Scanner;

class Mangu
{
  private String adm_no, name;
  private double marks;

  public void check()
  {
   if(marks >= 50)
      System.out.println("This Student has passed.");
    else
      System.out.println("This student has failed.");
  }

  public void dataout()
  {
    System.out.println("Adm No :> " + adm_no);
    System.out.println("Name   :> " + name);
    System.out.println("Marks  :> " + marks);
```

```
  }

  public void set(String adm,String the_name,double mks)
  {
    adm_no = adm;
    name = the_name;
    marks = mks;
  }

  public void datain()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the student's admission number: ");
    adm_no = myScanner.nextLine();
    System.out.print("Enter his/her name: ");
    name = myScanner.nextLine();
    System.out.print("Enter the marks he/she scored: ");
    marks = myScanner.nextDouble();
  }
}

class MySchool
{
  public static void main(String args[])
  {
    Mangu stud1 = new Mangu();
    Mangu stud2 = new Mangu();
    Mangu stud3 = new Mangu();

    stud1.set("DIT-001-3456/2000","Antony Kirui",89);
    stud2.set("CIT-001-4256/2000","Jacinta Wairimu",36);

    System.out.println("Analysis for the first student.");
    System.out.println("------------------------------------------------");
    stud1.dataout();
    stud1.check();

    System.out.println("\nAnalysis for the second student.");
    System.out.println("------------------------------------------------");
    stud2.dataout();
    stud2.check();

    System.out.println("\nData input for the third student.");
    System.out.println("------------------------------------------------");
    stud3.datain();
```

```
    System.out.println("\nAnalysis for the third student.");
    System.out.println("-----------------------------------------------");
    stud3.dataout();
    stud3.check();
  }
}
```

The program above should be saved as MySchool.java

The class "MyClass" can be explicitly declared a public but "Mangu" cannot.

public class MyClass is okay but **public class Mangu** is not. If you explicitly declare "Mangu" as public, you will get an error message stating: "error: class Mangu is public, should be declared in file named Mangu.java".

Example 4

```
class Motorcycle
{
  private String make;
  private String colour;
  private boolean engineState;

  public void setValues(String theMake,String theColour,boolean theState)
  {
    make = theMake;
    colour = theColour;
    engineState = theState;
  }

  public void setValues(String theMake,String theColour)
  {
    make = theMake;
    colour = theColour;
  }

  public void startEngine()
  {
     if (engineState == true)
         System.out.println("The engine is already on.");
     else
     {
         engineState = true;
         System.out.println("The engine is now on.");
     }
  }

  public void showAtts()
```

```
  {
    System.out.println("This motorcycle is a " + colour + " " + make);
    if (engineState == true)
       System.out.println("The engine is on.");
    else
          System.out.println("The engine is off.");
  }
}

class MyBike
{
  public static void main (String args[])
  {
    Motorcycle m = new Motorcycle();

    m.setValues("Yamaha RZ350","yellow");

    System.out.println("\nCalling showAtts...");
    System.out.println("-------------------------------------------------");
    m.showAtts();

    System.out.println("\nStarting engine...");
    System.out.println("-------------------------------------------------");
    m.startEngine();

    System.out.println("\nCalling showAtts...");
    System.out.println("-------------------------------------------------");
    m.showAtts();

    System.out.println("\nStarting engine...");
    System.out.println("-------------------------------------------------");
    m.startEngine();
  }
}
```

Example 5

```
import java.util.Scanner;

class Circle
{
  private double radius, area, circum;

  public void getData()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the radius of the triangle: ");
```

```
    radius = myScanner.nextDouble();
  }

  public void compute()
  {
    final double PI = 22.0/7;

    area = PI * radius * radius;
    circum = 2 * PI * radius;
  }

  public void giveOutput()
  {
    System.out.println("Radius of the circle = " + radius);
    System.out.println("Area = " + area);
    System.out.println("Circumference = " + circum);
  }
}

class CircleProgram
{
  public static void main(String args[])
  {
    Circle circ1 = new Circle();
    Circle circ2 = new Circle();

    System.out.println("\nData entry for circle 1:");
    System.out.println("---------------------------------------");
    circ1.getData();
    circ1.compute();

    System.out.println("\nData entry for circle 2:");
    System.out.println("---------------------------------------");
    circ2.getData();
    circ2.compute();

    System.out.println("\nOutput for circle 1:");
    System.out.println("---------------------------------------");
    circ1.giveOutput();

    System.out.println("\nOutput for circle 2:");
    System.out.println("---------------------------------------");
    circ2.giveOutput();
  }
}
```

Example 6

```
import java.util.Scanner;

class Triangle
{
  private double base, height, hypotenuse;
  private double area, perimeter;

  public void input()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the base of the triangle: ");
    base = myScanner.nextDouble();
    System.out.print("Enter the height of the triangle: ");
    height = myScanner.nextDouble();
  }

  public void calculate()
  {
    area = 1.0/2 * base * height;
    hypotenuse = Math.sqrt(Math.pow(base,2) +  Math.pow(height,2));
    perimeter = base + height + hypotenuse;
  }

  public void output()
  {
    System.out.println("Base = " + base + "\nHeight = " + height);
    System.out.println("Area = " + area);
    System.out.println("Perimeter = " + perimeter);
  }
}

class TriangleProgram
{
  public static void main(String args[])
  {
    Triangle tri = new Triangle();

    System.out.println("\nData entry for triangle:");
    System.out.println("----------------------------------------");
    tri.input();
    tri.calculate();

    System.out.println("\nOutput for the triangle:");
    System.out.println("----------------------------------------");
```

```
    tri.output();
  }
}
```

Working with Members

The *members* of a class are the fields and methods defined in the class body.

The following sections describe the basics of working with fields and methods in classes.

Fields

A *field* is a variable that's defined in the body of a class, outside of any of the class' methods. Fields, which are also called *class variables,* are available to all the methods of a class. In addition, if the field specifies the public keyword, the field is visible outside of the class. If you don't want the field to be visible outside of the class, use the private keyword instead.

Fields are defined the same as any other Java variable, but can have a modifier that specifies whether the field is public or private. Below are some examples of public field declarations:

public int marks = 0;

public String name;

public Student stud;

To create a private field, specify private instead of public:

private int initial_position = 0;

private String message = "";

Fields can also be declared as final. This represents a constant. See example below

public final int MAX_UNITS = 8;

The value of a final field can't be changed once it has been initialized.

Note: Spelling final field names with all capital letters is customary (but not required).

Methods

To declare a method that's available to users of your class, add the public keyword to the method declaration:

```
public double getArea()
{
  return area;
}
```

To create a private method that can be used within the class but isn't visible outside of the class, use the private keyword:

```
private void calculateDiameter()
{
// code to calculate the diameter goes here
```

}

Understanding visibility

Both fields and methods can use the public or private keywords to indicate whether or not the field or method can be accessed from outside of the class. This is called the *visibility* of the field or method.

The combination of all the members that have public access is sometimes called the *public interface* of your class. These members are the only means that other objects have to communicate with objects created from your class.

As a result, one should carefully consider which public fields and methods the class declares.

The term *expose* is sometimes used to refer to the creation of public fields and methods. For example, if a class has a public method named calculate, one can say that the class exposes the calculate method. That simply means that the method is available to other classes.

One can use private fields and methods within a class but not from other classes. They're used to provide implementation details that may be crucial to the operation of the class, but that shouldn't be exposed to the outside world. Private fields and methods are sometimes called *internal members,* because they're available only from within the class.

Getters and Setters

One of the basic goals of object-oriented programming is to hide the implementation details of a class inside the class while carefully controlling what aspects of the class are exposed to the outside world. As a general rule, one should avoid creating public fields. All the fields should be made private.

One can then selectively grant access to the data those fields contain by adding special methods called *accessors* to the class.

There are two types of accessors: A *get accessor* (also called a *getter*) is a method that retrieves a field value, while a *set accessor* (*setter*) is a method that sets a field value. These methods are usually named get*FieldName* and set*FieldName,* respectively. For example, if the field is named age, the getter and setter methods are named getAge and setAge.

The example below illustrates the use of a get accessor and a set accessor

```
public class Person
{
  private int age;

  public int getAge()
  {
    return age;
  }

  public void setAge(int ag)
```

```
  {
    age = ag;
  }
}
```

Here, the age field is declared as private, so it can't be accessed directly. Instead, it can be accessed only through the methods getAge and setAge.

Creating a class with accessors rather than simple public fields has several benefits:

i.) You can create a read-only property by providing a get accessor but not a set accessor. Then, other classes can retrieve the property value, but can't change it.

ii.) Instead of storing the property value in a private field, you can calculate it each time the get accessor method is called. For example, suppose you have a class named Order that includes fields named unitPrice and quantityOrdered. This class might also contain a getOrderTotal method that looks like this:

```
public double getOrderTotal()
{
return unitPrice * quantityOrdered;
}
```

Here, instead of returning the value of a class field, the get accessor calculates the value to be returned.

iii.) You can protect the class from bad data by validating data in a property set accessor and either ignoring invalid data or throwing an exception if invalid data is passed to the method. For example, suppose you have a set accessor for an int property named age whose value can be from 0 to 100. Here's a set accessor that prevents the age property from being set to an incorrect value:

```
public void setAge(int ag)
{
  if (ag < 0)
    age = 0;
  else if (ag > 100)
    age = 100;
  else
    age = ag;
}
```

Here, if the setAge method is called with a value less than zero, the age is set to zero. Likewise, if the value is greater than 100, the age is set to 100.

Note/Caution

The use of accessors an example of a design pattern that's commonly used by Java programmers. The *Accessor pattern* is designed to provide a consistent way to set or retrieve the value of class fields without having to expose the fields themselves to the outside world.

Most Java programmers quickly learn that one of the basic guidelines of object-oriented programming is to avoid public fields. Unfortunately, they often respond to this guideline by making all fields private, and then providing get and set accessors for **every field** whether they need them or not.

The point of making fields private is so that you can carefully control access to them. If you blindly create accessors for **all** your fields, you may as well just make the fields public.

Instead, carefully consider which fields really should be accessible to the outside world, and provide accessors **only for those fields** that really need them.

Example

The example below illustrates a complete program that makes use of accessor methods.

```
import java.util.Scanner;

class Mtu
{
  private int age;

  public int getAge()
  {
    return age;
  }

  public void setAge(int ag)
  {
    age = ag;
  }
}

public class AgeProgram
{
  public static void main(String args[])
  {
    Scanner myScanner = new Scanner(System.in);

    Mtu m1 = new Mtu();
    Mtu m2 = new Mtu();
    int theAge1, theAge2;

    m1.setAge(23);

    System.out.print("\nEnter the age for the second person:> ");
    theAge2 = myScanner.nextInt();
    m2.setAge(theAge2);
```

```
    theAge1 = m1.getAge();

    System.out.println("\nThe first person is " + theAge1 + " years old.");
    System.out.println("The second person is " + m2.getAge() + " years old.");
  }
}
```

Overloading Methods

Method overloading is having two or more methods in the same class having the same name (but different signatures – number and/or type of parameters).

Method overloading creates flexibility into classes. With overloading, one can anticipate different ways someone might want to invoke an object's functions, and then provide overloaded methods for each alternative.

The basic rule when creating overloaded methods is that every method must have a unique signature. A method's *signature* is the combination of its name and the number and types of parameters it accepts.

Note:

There are two things that are ***not*** a part of a method's signature. These are:

i.) The method's return type: You can't code two methods with the same name and parameters but with different return types.

ii.) The names of the parameters: All that matters to the method signature are the types of the parameters and the order in which they appear.

Thus, the following two methods have the same signature:

```
double myMethod(double x, boolean y)
double myMethod(double param1, boolean param2)
```

Example

In the example below

```
class OverloadNum
{
  int first, second, third;

  public void setData(int x, int y)
  {
    first = x;
    second = y;
  }

  public void setData(int x,int y,int z)
  {
    first = x;
    second = y;
    third = z;
```

```
  }

  public void output()
  {
    System.out.println("First Number = " + first);
    System.out.println("Second Number = " + second);
    System.out.println("Third Number = " + third);
  }
}

class OverloadExample
{
  public static void main(String args[])
  {
    OverloadNum set1 = new OverloadNum();
    OverloadNum set2 = new OverloadNum();

    set1.setData(12,52);
    set2.setData(89,7,76);

    System.out.println("\nData output for set 1:");
    System.out.println("------------------------");
    set1.output();

    System.out.println("\nData output for set 2:");
    System.out.println("------------------------");
    set2.output();
  }
}
```

Constructors

A constructor is a method that is called automatically when an instance of a class is declared and its function is to initialize some or the instance variables of the class.

Constructors have two main characteristics

i.) They have the same name as the class.

ii.) They have no return type.

The most common reason for coding a constructor is to provide initial values for class fields when you create the object. For example, suppose you have a class named Lecturer that has fields named firstName and lastName. You can create a constructor for the Lecturer class as shown below:

```
public Lecturer(String fName, String lName)
{
        firstName = fName;
        lastName = lName;
```

```
}
```

You can create an instance of the Lecturer class by calling this constructor:

Lecturer lect1 = new Lecturer("John", "Maina");

A new Lecturer object with initial values John Maina is created.

Like other methods, constructors can be overloaded and in most cases are. That is, you can provide more than one constructor for a class, provided each constructor has a unique signature. For example, here's another constructor for the Lecturer class:

```
public Lecturer(String fName, String lName,int years)
{
        firstName = fName;
        lastName = lName;
        yearsOfExperince = years
}
```

This constructor lets you create a Lecurer object with additional information besides the first and last names:

Lecturer lect1 = new Lecturer("John", "Maina",42);

NB: You can create a Lecturer object and initialize it only with the names or with the names and age. The appropriate constructor will be used [automatically].

Default Constructors

A default constructor is a constructor that does not accept parameters.

If you introduce constructors in your class you **must** also include a default constructor. If you don't, it will not be possible to declare objects without initializing.

Example 1

```
import java.util.Scanner;

class Lecturer
{
  private String firstName, lastName;
  private int yearsOfExperince;

  public Lecturer(String fName, String lName,int years)
  {
    firstName = fName;
    lastName = lName;
    yearsOfExperince = years;
  }

  public Lecturer(String fName, String lName)
  {
    firstName = fName;
    lastName = lName;
```

```
  }

  public Lecturer()//Default Constructor
  {
    firstName = "";
    lastName = "";
    yearsOfExperince = 0;
  }

  public void input()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the lecturer's first name:> ");
    firstName = myScanner.nextLine();
    System.out.print("Enter his/her last name:> ");
    lastName = myScanner.nextLine();
    System.out.print("Enter his/her experience (number of years):> ");
    yearsOfExperince = myScanner.nextInt();
  }

  public void output()
  {
    System.out.println("First Name: " + firstName);
    System.out.println("Last Name: " + lastName);
    System.out.println("Years of Experience: " + yearsOfExperince);
  }
}

class LecturerDetails
{
  public static void main(String args[])
  {
    Lecturer lect1 = new Lecturer("John", "Maina",42);
    Lecturer lect2 = new Lecturer("Edwin", "Muruka");
    Lecturer lect3 = new Lecturer();

    System.out.println("\nData output for lecturer 1:");
    System.out.println("----------------------------------------");
    lect1.output();

    System.out.println("\nData output for lecturer 2:");
    System.out.println("----------------------------------------");
    lect2.output();

    System.out.println("\nData output for lecturer 3:");
```

```
    System.out.println("----------------------------------------");
    lect3.output();

    System.out.println("\nData input for lecturer 3:");
    System.out.println("----------------------------------------");
    lect3.input();

    System.out.println("\nData output for lecturer 3 [Second Time]:");
    System.out.println("----------------------------------------");
    lect3.output();
  }
}
```

Example 2

```
import java.util.Scanner;

class Rectangle
{
  private double width, length, area, perimeter;

  public Rectangle(double wid,double len)
  {
    width = wid;
    length = len;
  }

  public Rectangle()
  {
    width = 1;
    length = 1;
  }

  public void datain()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the width of the rectangle -> ");
    width = myScanner.nextDouble();
    System.out.print("Enter the length of the rectangle -> ");
    length = myScanner.nextDouble();
  }

  public void calculate()
  {
    area = width * length;
    perimeter = 2 * (width + length);
  }
```

```
  public void dataout()
  {
    System.out.println("Width = " + width);
    System.out.println("Length = " + length);
    System.out.println("Area = " + area);
    System.out.println("Perimeter = " + perimeter);
  }
}

class MyRectangle
{
  public static void main(String args[])
  {
    Rectangle rect1 = new Rectangle(6,10);
    Rectangle rect2 = new Rectangle();

    System.out.println("\nOutput for rectangle 1:");
    System.out.println("----------------------------------------");
    rect1.calculate();
    rect1.dataout();

    System.out.println("\nOutput for rectangle 2:");
    System.out.println("----------------------------------------");
    rect2.calculate();
    rect2.dataout();

    System.out.println("\nInput for rectangle 1:");
    System.out.println("----------------------------------------");
    rect1.datain();
    rect1.calculate();

    System.out.println("\nOutput for rectangle 1 [Second Time]:");
    System.out.println("----------------------------------------");
    rect1.dataout();
  }
}
```

Calling other constructors

A constructor can call another constructor of the same class by using the special keyword **this** as a method call. This technique is commonly used when you have several constructors that build on each other.

Below is an example example:

```
public class Lecturer
{
  private String firstName, lastName;
```

```
  private int yearsOfExperince;

  public Lecturer(String fName, String lName)
  {
    firstName = fName;
    lastName = lName;
  }
  public Lecturer(String fName, String lName,int years)
  {
    this(fName, lName);
    yearsOfExperince = years;
  }
}
```

Here, the second constructor calls the first constructor to set the firstName and lastName fields. Then, it sets the yearsOfExperince field.

There are a few restrictions on how to use the this keyword as a constructor call:

i.) You can call another constructor only in the very first statement of a constructor. Thus, the following will not compile:

```
public Lecturer(String fName, String lName,int years)
{
    yearsOfExperince = years;
    this(fName, lName);          //Error. Will not compile

}
```

ii.) Each constructor can call only one other constructor. However, you can chain constructors together. For example, if a class has three constructors, the first constructor can call the second one, which in turn calls the third one.

iii.) You can't create loops where constructors call each other. For example, here's a class that won't compile:

```
class MyClass
{
        private String firstString;
        private String secondString;

        public MyClass(String first, String second)
        {
                this(first);
                secondString = second;
        }
        public MyClass(String first)
        {
                this(first, "DEFAULT"); // error: will not compile
        }
}
```

The first constructor starts by calling the second constructor, which calls the first constructor. The compiler complains that this error is a recursive constructor invocation and refuses to compile the class.

If you don't explicitly call a constructor in the first line of a constructor, Java inserts code that automatically calls the default constructor of the base class – that is, the class that this class inherits.

Example

```
import java.util.Scanner;

class Lecturer
{
  private String firstName, lastName;
  private int yearsOfExperince;

  public Lecturer(String fName, String lName)
  {
    firstName = fName;
    lastName = lName;
  }

  public Lecturer(String fName, String lName,int years)
  {
    this(fName,lName);
    yearsOfExperince = years;
  }

  public Lecturer()//Default Constructor
  {
    this("","",0);
  }

  public void input()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the lecturer's first name:> ");
    firstName = myScanner.nextLine();
    System.out.print("Enter his/her last name:> ");
    lastName = myScanner.nextLine();
    System.out.print("Enter his/her experience (number of years):> ");
    yearsOfExperince = myScanner.nextInt();
  }

  public void output()
  {
```

```
    System.out.println("First Name: " + firstName);
    System.out.println("Last Name: " + lastName);
    System.out.println("Years of Experience: " + yearsOfExperince);
  }
}

class LecturerDetailsTwo
{
  public static void main(String args[])
  {
    Lecturer lect1 = new Lecturer("John", "Maina",42);
    Lecturer lect2 = new Lecturer("Simon", "Makonde");
    Lecturer lect3 = new Lecturer();

    System.out.println("\nData output for lecturer 1:");
    System.out.println("----------------------------------------");
    lect1.output();

    System.out.println("\nData output for lecturer 2:");
    System.out.println("----------------------------------------");
    lect2.output();

    System.out.println("\nData output for lecturer 3:");
    System.out.println("----------------------------------------");
    lect3.output();

    System.out.println("\nData input for lecturer 3:");
    System.out.println("----------------------------------------");
    lect3.input();

    System.out.println("\nData output for lecturer 3 [Second Time]:");
    System.out.println("----------------------------------------");
    lect3.output();
  }
}
```

More uses for this

As discussed above, you can use the this keyword in a constructor to call another constructor for the current class. You can also use this in the body of a class constructor or method to refer to the current object – that is, the class instance for which the constructor or method has been called.

The this keyword is usually used to qualify references to instance variables of the current object. For example:

```
public Lecturer(String fName, String lName)
{
    this.firstName = fName;
```

```
    this.lastName = lName;
}
```

Here, this isn't really necessary because the compiler can tell that lastName and firstName refer to class variables. However, suppose you use lastName and firstName as the parameter names for the constructor:

```
public Lecturer(String firstName, String lastName)
{
    this.firstName = firstName;
    this.lastName = lastName;
}
```

Here, the this keywords are required to distinguish between the parameters named lastName and firstName and the instance variables with the same names.

You can also use this in a method body. For example:

```
public String getFullName()
{
        return this.firstName + " " + this.lastName;
}
```

Because this example has no ambiguity, 'this' isn't really required. However, many programmers like to use 'this' even when it isn't necessary because it makes it clear that you're referring to an instance variable.

Sometimes, you use the this keyword all by itself to pass a reference to the current object as a method parameter. For example, you can print the current object to the console by using the following statement:

```
System.out.println(this);
```

The println method calls the object's toString method to get a string representation of the object, and then prints it to the console. By default, toString prints the name of the class that the object was created from and the object's hash code. If you want the println method to print something more meaningful, provide a toString method of your own for the class.

Class Variables

Class variables are variables whose value applies to the class and all its instances.

With instance variables, each object gets a new copy of the instance variables the class defines and you can change these values for one object without affecting the other objects.

With class variables, there is only one copy of that variable and every instance of that class has access to that value. If this value is changed, it changes for ALL the instance (objects) of that class.

We use the **static** keyword to define class variable.

Example.

```
class Family
{
    private static String surname = “Maina”;
    private String name;
    private int age;
}
```

Surname has been declared as a class variable and as a result, all objects of the class Family will have the same value for this variable.

Example

A program that accepts names of family members. It illustrates the use of ***static*** to come up with class variables.

```
//program to accept names of family members
//illustrates the use of static to come up with class variables
import java.util.Scanner;

class Family
{
  private static String surname;
  private String firstName;
  private int yearOfBirth;

  public static void getSurname()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the family's surname:> ");
    surname = myScanner.nextLine();
  }

  public void datain()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the name:> ");
    firstName = myScanner.nextLine();
    System.out.print("Enter the year of birth:> ");
    yearOfBirth = myScanner.nextInt();
  }

  public void dataout()
  {
    System.out.println("Name: " + firstName + " " + surname);
    System.out.println("Year of Birth: " + yearOfBirth);
```

```
  }
}

class YourFamily
{
  public static void main(String args[])
  {
    Family father = new Family();
    Family mother = new Family();
    Family you = new Family();

    System.out.println("\nFamily Name Input:");
    System.out.println("-----------------------------------");
    Family.getSurname();

    System.out.println("\nData input for your father:");
    System.out.println("-----------------------------------");
    father.datain();

    System.out.println("\nData input for your mother:");
    System.out.println("-----------------------------------");
    mother.datain();

    System.out.println("\nData input for you:");
    System.out.println("-----------------------------------");
    you.datain();

    System.out.println("\nFather's Data:");
    System.out.println("-----------------------------------");
    father.dataout();

    System.out.println("\nMother's Data:");
    System.out.println("-----------------------------------");
    mother.dataout();

    System.out.println("\nYour Data:");
    System.out.println("-----------------------------------");
    you.dataout();
  }
}
```

To access class variables you can use the name of any object of that class or the name of the class itself. E.g. in the above example, we can use father.getSurname(), mother.getSurname(), you.getSurname() or simply Family.getSurname() to read the surname and the end result will be the same. It is less confusing through to simply use the name of the class e.g. Family.getSurname()

Working with statics

A *static method* is a method that is not associated with an instance of a class (an object). Instead, the method belongs to the class itself. As a result, you can call the method without first creating a class instance. In the above example getSurname() is an example of a static method.

Understanding Static Fields and Methods

"static" is used to describe a special type of field or method that isn't associated with a particular instance of a class. Instead, static fields and methods are associated with the class itself.

That means you don't have to create an instance of the class to access a static field or methods. Instead, you access a static field or method by specifying the class name, not a variable that references an object.

Working with Static Fields

A *static field* is a field whose value applies to the class and all its instances. It is declared with the static keyword as shown below:

private static int noOfUnits;

"static" fields are normally used to provide constants or other values that aren't related to class instances. For example if all students taking a certain course are taking the same number of units, it would be easier/more efficient to declare noOfUnits as a static field so that the value can be entered only once but applied to all instances of the class.

Note that you can't use the static keyword within a class method. Thus, the following code won't compile:

```
static private void MyMethod()
{
        static int x;
}
```

In other words, instance variables/fields can be static, but local variables cannot.

You can provide an initial value for a static field:

private static String surname = "Maina";

Using Static Methods

A *static method* is a method associated with the class itself, not with any particular object created from the class. As a result, you don't have to create an object from a class before you can use static methods defined by the class. They are declared with the static keyword.

One of the basic rules of working with static methods is that you can't access a non-static method or field from a static method. That's because the static method doesn't have an instance of the class to use to reference instance methods or fields. For example, the following code won't compile:

```
class Family
{
  private String firstName;  //An instance variable

  public static void setNames()
  {
    firstName = "John”; //Error. Will not compile.
  }
```

Getting the Name of the Class an Object Belongs to

If you want to get the name of the class an object belongs to use the technique below.

object.name.getClass().getName();

For example if you have an object called Kamau and you want to know to which class it belongs use:-

kamau.getClass().getname();

Instanceof Operator

instanceof operator has two operands: an object on the left and the name of the class right. The expression returns true or false based on whether the object is an instance of the named class or any of its subclasses e.g. given: -

Family father = new Family ();

father instanceof Family returns true

Example

```
import java.util.Scanner;

class Worker
{
  private String name,department;

  public void input()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the employee's name:> ");
    name = myScanner.nextLine();
    System.out.print("Enter his/her department:> ");
    department = myScanner.nextLine();
  }

  public void dataout()
  {
    System.out.println("Name: " + name);
    System.out.println("Department: " + department);
  }
```

```
}

class NairobiLTD
{
  public static void main(String args[])
  {
    Worker wk1 = new Worker();
    boolean resp;

    System.out.println("\nThe object belongs to a class called " +
wk1.getClass().getName());

    resp = wk1 instanceof Worker;
    System.out.println("Does the object belongs to a class called Worker? " + resp);
  }
}
```

INHERITANCE

Inheritance is defining a new class based on an already existing class such that the new class acquires all the methods and variables of the existing class (the class it is inheriting from)

The child class can use the fields or methods it inherits as is, or it can override them to provide its own versions. In addition, the child class can add fields or methods of its own.

Superclass/Parent Class/Base Class

This is a class whose methods and variables are acquired by another class through the process of inheritance (i.e. the class that is inherited from by another class)

Subclass/Child Class/Derived Class

This is a class that acquires the methods and variables of another class through the process of inheritance (i.e. the class that inherits from another class)

Subclassing

Subclassing is the process of creating a new class that inherits from some other already existing class.

Facts about Inheritance:

i.) A derived class automatically takes on all the behavior and attributes of its base class. Thus, if you need to create several different classes to describe types that aren't identical but have many features in common, you can create a base class that defines all the common features. Then, you can create several derived classes that inherit the common features.

ii.) A derived class can add features to the base class it inherits by defining its own methods and fields. This is one way a derived class distinguishes itself from its base class.

iii.) A derived class can also change the behavior provided by the base class. For example, a base class (with a method called play) may provide that all (or some) classes derived from it have a method named play, but each class is free to provide its own implementation of the play method. In this case, all (or some) classes that extend the base class provide their own implementation of the play method.

iv.) Inheritance is best used to implement *is-a-type-of* relationships. For example: Football is a type of Game; A Matatu is a type of vehicle; A PartTimeStudent is a type of Student. In each case, a particular kind of object is a specific type of a more general category of objects.

Consider various types of vehicles. Personal Cars and Matatus are two distinct types of vehicles. If you're writing software that represented vehicles, you could start by creating a class called **Vehicle** that would describe the features that are common to all types of vehicles, such as registration number, colour, year of manufacture etc., and the ability to perform actions such as driving, stopping, turning, or crashing.

A Matatu is a type of vehicle that further refines the **Vehicle** class. The **Matatu** class would inherit the **Vehicle** class, so it would have registration number, colour, year of manufacture etc., and the ability to perform actions such as driving, stopping, turning, or crashing. In addition, it would have features that differentiate it from other types of vehicles. For example, it has a route, no of passengers etc.

A personal car is also a type of vehicle. The **PersonalCar** class would inherit the **Vehicle** class, so it too would have registration number, colour, year of manufacture etc., and the ability to perform actions such as driving, stopping, turning, or crashing. Plus it would have some features of its own.

How Inheritance Works

When you create a new instance of a class (an object) you get a "slot" for each variable defined in the current class and for each variable defined in all its super classes.

Methods operate similarly: New objects have access to all methods in its class and its super classes but method definitions are chosen dynamically when a method is called.

That is, if you call a method on a particular object, Java first checks the object's class for the definition of that method. If it is not defined in the object's class it looks for it in that class's superclass and so on up the chain until the method is found.

When a subclass defines a method that has the same signature (name, number and type of arguments) as a method defined in a superclass, the method definition that is found first (starting at the bottom and working up toward the top of the hierarchy) is the one that is actually executed. Therefore you can intentionally define a method in a super class, which then "hides" the super class's method. This is called overriding a method.

Creating Subclasses

The basic procedure for creating a subclass is simple. You just use the "**extends"** keyword on the declaration for the subclass. The basic format of a class declaration for a class that inherits a base class is as shown below:

[public] class *ClassName* extends *BaseClass*
{
	// class body goes here
}

For example, suppose you have a class named Ball that defines a basic ball, and you want to create a subclass named BouncingBall that adds the ability to bounce.

```
public class BouncingBall extends Ball
{
	// methods and fields that add the ability to bounce
	// to a basic Ball object:
	public void bounce( )
	{
		// the bounce method
	}
}
```

Example 1

In the example below, the base class is “Rectangle” and the base class, which inherits the members of “Rectangle” is “Cube”.

```
import java.util.Scanner;

class Rectangle
{
  protected double width,length,area;

  public void data_in()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the width: ");
    width = myScanner.nextDouble();
    System.out.print("Enter the length: ");
    length = myScanner.nextDouble();
  }

  public void data_calculate()
  {
    area = width * length;
  }

  public void data_out()
  {
    System.out.println("The area of the rectangle is " + area);
  }
}
```

```
import java.util.Scanner;

class Cube extends Rectangle
{
  private double height, volume;

  public void in()
  {
    Scanner myScanner = new Scanner(System.in);

    data_in();

    System.out.print("Enter the height: ");
    height = myScanner.nextDouble();
  }

  public void compute()
  {
    data_calculate();
    volume = area * height;
  }

  public void out()
  {
    System.out.println("The volume of the cube is " + volume);
  }
}

class Shapes
{
  public static void main(String args[])
  {
    Rectangle rect1 = new Rectangle();
    Cube cub1 = new Cube();

    System.out.println("\nInput for the rectangle:");
    System.out.println("----------------------------------------");
    rect1.data_in();
    rect1.data_calculate();

    System.out.println("\n\nInput for the Cube:");
    System.out.println("----------------------------------------");
    cub1.in();
    cub1.compute();
```

```
    System.out.println("\n\nAnalysis for the rectangle:");
    System.out.println("----------------------------------------");
    rect1.data_out();

    System.out.println("\n\nAnalysis for the Cube:");
    System.out.println("----------------------------------------");
    cub1.out();
  }
}
```

Example 2

In the example below, the base class is "Vehicle" and the base class, which inherits the members of "Vehicle" is "Matatu".

```
import java.util.Scanner;

class Vehicle
{
  protected String make, countryOfManuf;
  protected int yearOfManuf;

  public void v_input()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the make of the Vehicle: ");
    make = myScanner.nextLine();
    System.out.print("Enter the country of manufacture: ");
    countryOfManuf = myScanner.nextLine();
    System.out.print("Enter its year of manufacture: ");
    yearOfManuf = myScanner.nextInt();
  }

  public void v_output()
  {
    System.out.println("Make of the Vehicle: " + make);
    System.out.println("Country of Manufacture: " + countryOfManuf);
    System.out.println("Year of Manufacture: " + yearOfManuf);
  }
}

import java.util.Scanner;

class Matatu extends Vehicle
{
  private String route;
  private int speed_limit,passengers;
```

```
  public void m_input()
  {
    Scanner myScanner = new Scanner(System.in);

    v_input();

    System.out.print("Enter the matatu's route: ");
    route = myScanner.nextLine();
    System.out.print("Enter its speed limit: ");
    speed_limit = myScanner.nextInt();
    System.out.print("Enter the maximum number of passengers: ");
    passengers = myScanner.nextInt();
  }

  void m_output()
  {
        v_output();
        System.out.println("Route: " + route);
        System.out.println("Speed Limit: " + speed_limit + " km/h");
        System.out.println("No of passengers: " + passengers);
  }
}

class Magari
{
  public static void main(String args[])
  {
    Vehicle personal = new Vehicle();
    Matatu mat = new Matatu();

    System.out.println("\nData entry for personal car: ");
    System.out.println("-----------------------------------------");
    personal.v_input();

    System.out.println("\nData entry for the matatu: ");
    System.out.println("-----------------------------------------");
    mat.m_input();

    System.out.println("\nAnalysis for the personal car: ");
    System.out.println("-----------------------------------------");
    personal.v_output();

    System.out.println("\nAnalysis for the matatu: ");
    System.out.println("-----------------------------------------");
    mat.m_output();
```

```
  }
}
```

Example 3

In the example below, the base class is "TheStudent" and the base classes, which inherit the members of "TheStudent" are "PartTimeStudent" and "FullTimeStudent".

```
import java.util.Scanner;

class TheStudent
{
  protected String admNo, sName, course;

  public void sInput()
  {
    Scanner myScanner = new Scanner(System.in);

    System.out.print("Enter the admission number of the student: ");
    admNo = myScanner.nextLine();
    System.out.print("Enter his/her name: ");
    sName = myScanner.nextLine();
    System.out.print("Enter the course he/she is doing: ");
    course = myScanner.nextLine();
  }

  public void sOutput()
  {
    System.out.println("Admission Number: " + admNo);
    System.out.println("Student's name: " + sName);
    System.out.println("Course: " + course);
  }
}

import java.util.Scanner;

class PartTimeStudent extends TheStudent
{
  private String employerName, dateOfEmployment, employmentMode;

  public void ptsInput()
  {
    Scanner myScanner = new Scanner(System.in);

    sInput();

    System.out.print("Enter his/her employer: ");
    employerName = myScanner.nextLine();
```

```
    System.out.print("Enter his/her date of employment: ");
    dateOfEmployment = myScanner.nextLine();
    System.out.print("Enter his/her employment mode [permanent/contract/casual]: ");
    employmentMode = myScanner.nextLine();
  }

  public void ptsOutput()
  {
    sOutput();

    System.out.println("Name of Employer: " + employerName);
    System.out.println("Date of Employment: " + dateOfEmployment);
    System.out.println("Employment Mode: " + employmentMode);
  }
}

import java.util.Scanner;

class FullTimeStudent extends TheStudent
{
  private String guardiansName, guardiansTelNo;

  public void ftsInput()
  {
    Scanner myScanner = new Scanner(System.in);

    sInput();

    System.out.print("Enter the name of the student's guardian: ");
    guardiansName = myScanner.nextLine();
    System.out.print("Enter guardian's telephone number: ");
    guardiansTelNo = myScanner.nextLine();
  }

  public void ftsOutput()
  {
    sOutput();

    System.out.println("Guardian's Name: " + guardiansName);
    System.out.println("Guardians Telephone Number: " + guardiansTelNo);
  }
}

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

```
  {
    PartTimeStudent stud1 = new PartTimeStudent();
    FullTimeStudent stud2 = new FullTimeStudent();

    System.out.println("\nData entry for Student 1 [Part Time Student]: ");
    System.out.println("-------------------------------------------");
    stud1.ptsInput();

    System.out.println("\nData entry for Student 2 [Full Time Student]: ");
    System.out.println("-------------------------------------------");
    stud2.ftsInput();

    System.out.println("\nAnalysis for Student 1 [Part Time Student]: ");
    System.out.println("-------------------------------------------");
    stud1.ptsOutput();

    System.out.println("\nAnalysis for Student 2 [Full Time Student]: ");
    System.out.println("-------------------------------------------");
    stud2.ftsOutput();
  }
}
```

Overriding Methods

Overriding a method is creating a method in a subclass that has the same signature (name, number and type of arguments) as a method in a superclass. That new method then hides the super class's method.

Method overriding lets you modify the behavior of a base class to suit the needs of the subclass.

Example 1

In the example below, "Second" extends (inherits from) "First" and overrides the "greetings()" method.

```
class First
{
  public void greetings()
  {
    System.out.println("How are you doing my friend?");
  }

  public void bye()
  {
    System.out.println("Goodbye. Have a nice day.");
  }
}

class Second extends First
```

```
{
  public void comment()
  {
    System.out.println("Today is a very hot day!!!");
  }

  public void greetings()//overriding
  {
    System.out.println("Habari yako.");
  }
}

class OverrideOne
{
  public static void main(String args[])
  {
    First obj1 = new First();
    Second obj2 = new Second();

    System.out.println("\nWorking with the first object");
    System.out.println("---------------------------------");
    obj1.greetings();
    obj1.bye();

    System.out.println("\nWorking with the second object");
    System.out.println("---------------------------------");
    obj2.greetings();
    obj2.comment();
    obj2.bye();
  }
}
```

Example 2

In the example below, "TheCube" extends "TheRectangle" and overrides "dataIn()", "calculate()" and "dataOut()". "TheCube" calls the [overridden] methods in the superclass using super.methodName() e.g. super.dataIn().

```
import java.util.Scanner;

class TheRectangle
{
  protected double width,length,area;

  public void dataIn()
  {
    Scanner myScanner = new Scanner(System.in);
```

```
    System.out.print("Enter the width: ");
    width = myScanner.nextDouble();
    System.out.print("Enter the length: ");
    length = myScanner.nextDouble();
  }

  public void calculate()
  {
    area = width * length;
  }

  public void dataOut()
  {
    System.out.println("The area of the rectangle is " + area);
  }
}

import java.util.Scanner;

class TheCube extends TheRectangle
{
  private double height, volume;

  public void dataIn()//override.
  {
    Scanner myScanner = new Scanner(System.in);

    super.dataIn();

    System.out.print("Enter the height: ");
    height = myScanner.nextDouble();
  }

  public void calculate()//override.
  {
    super.calculate();
    volume = area * height;
  }

  public void dataOut()//override.
  {
    System.out.println("The volume of the cube is " + volume);
  }
}

class TheShapes
```

```
{
  public static void main(String args[])
  {
    TheRectangle rect1 = new TheRectangle();
    TheCube cub1 = new TheCube();

    System.out.println("\nInput for the rectangle:");
    System.out.println("----------------------------------------");
    rect1.dataIn();
    rect1.calculate();

    System.out.println("\n\nInput for the Cube:");
    System.out.println("----------------------------------------");
    cub1.dataIn();
    cub1.calculate();

    System.out.println("\n\nAnalysis for the rectangle:");
    System.out.println("----------------------------------------");
    rect1.dataOut();

    System.out.println("\n\nAnalysis for the Cube:");
    System.out.println("----------------------------------------");
    cub1.dataOut();
  }
}
```

Single and Multiple Inheritance

Java's form of inheritance is called single inheritance. Single Inheritance means that each (Java) class can have only one superclass (although any given superclass can have multiple subclasses)

In other object oriented programming languages, such as C++, classes can have more than one superclass, and they inherit combined variables and methods from all those classes. This is called Multiple Inheritance

Because each Java class has only a single superclass, it only inherits variables and methods from that superclass and all its superclasses. To solve the problem that arises due to lack of multiple inheritance Java uses the concept of interfaces, which collect method names into one place and hen allows you to add these methods as a group to the various classes that need them.

Inheritance, Interfaces and Packages

An Interface

An interface is a collection of method names, without definitions, that can be added to classes to provide additional behaviour not included with those methods the class defined itself or inherited from its superclass.

Package

A package is a collection of classes and interfaces.

Public, Private and Protected Members

Public Members – Public members are members of a class that can be accessed everywhere the class is visible. They can be accessed by both members and non-members of the class.

Private Members – Private members are members of a class that can only be accessed by other members of the same class.

Protected Members – Protected members are members of class that can be accessed by other members of the same class and also by members of the subclasses.

Inheritance and Constructors

When you create an instance of a subclass, Java automatically calls the default constructor of the base class before it executes the subclass constructor.

Example

```
class Games
{
  public Games()
  {
    System.out.println("Hello from the Games constructor.");
  }
}

class FootBall extends Games
{
  public FootBall()
  {
    System.out.println("Hello from the FootBall constructor.");
  }
}

class InheritanceConstrutor
{
  public static void main(String args[])
  {
    FootBall manunited = new FootBall();
    //Other code goes here.
  }
}
```

When the above program is run, the output will be:

```
Hello from the Games constructor.
Hello from the FootBall constructor.
```

Explanation

The declaration *FootBall manunited = new FootBall();* calls the [default] constructor for the "FootBall" class. When the FootBall class constructor [any] is called the default constructor of the superclass [Games] is also **automatically** called.

If you want, you can explicitly call a base class constructor from a subclass by using the "super" keyword. Because Java automatically calls the default constructor for you, the only reason to do this is to call a constructor of the base class that uses a parameter(s).

```
import java.util.Scanner;

class Rectangle
{
  protected double width,length,area;

  public Rectangle(double wid, double len)
  {
    width = wid;
    length = len;
  }

  public Rectangle()
  {
    width = 1;
    length = 1;
  }

  //Other code goes here
}

import java.util.Scanner;

class Cube extends Rectangle
{
  private double height, volume;

  public cube(double wid, double len, double hgt)
  {
    super(wid, len);
    height = hgt;
  }

  public cube()
  {
    //Default constructor for Rectangle class will be called automatically.
    //Not necessary to call it.
    height = hgt;
  }
```

Factor to note when working with superclass constructors:

i.) If you use super to call the superclass constructor, it must the very first statement in the constructor.

ii.) If you don't explicitly call super, the compiler inserts a call to the default constructor of the base class.

iii.) If the superclass is itself a subclass, the constructor for its superclass is called in the same way. This continues all the way up the inheritance hierarchy, until you get to the Object class, which has no superclass.