Introduction to JAVA

C is procedural language.

whereas

Java is object-oriented language.

Procedure-Oriented Paradigm

- ✓ *Procedural programming means breaking your program into functions; where each function performs an specific task.*
- ✓ *You pass data back and forth to these functions, each function processes the data just a bit.*
- ✓ *Hence functions are the basis of the code and data plays a secondary role.*
- ✓ *So this might be called "Functions Oriented“ programming, or “Procedural programming.*

Object Oriented Paradigm

- ✓ *Emphasis is on data rather than procedure.*
- ✓ *Programs are divided into what are known as objects.*
- ✓ *Data structure are designed such that they characterize the objects.*
- ✓ *Methods that operate on the data of an object are tied together in the data structure.*
- ✓ *Data is hidden and cannot be accessed by external functions.*
- ✓ *Objects may communicate with each other through methods.*
- ✓ *New data and methods can be easily added whenever necessary.*
- ✓ *Follows bottom-up approach in program design.*

Why Java ???

Java is a programming language and computing platform first released by Sun Microsystems in 1995.

- ✓ *Java is fast, secure, and reliable.*
- ✓ *Write software on one platform and run it on virtually any other platform.*
- ✓ *Create programs to run within a Web browser and Web services.*

Why Java ???

- ✓ *Develop server-side applications for online forums, stores, polls, HTML forms processing, and more.*
- ✓ *Combine applications or services using the Java language to create highly customized applications or services.*
- ✓ *Write powerful and efficient applications for mobile phones, remote processors, low-cost consumer products, and practically any other device with a digital heartbeat.*

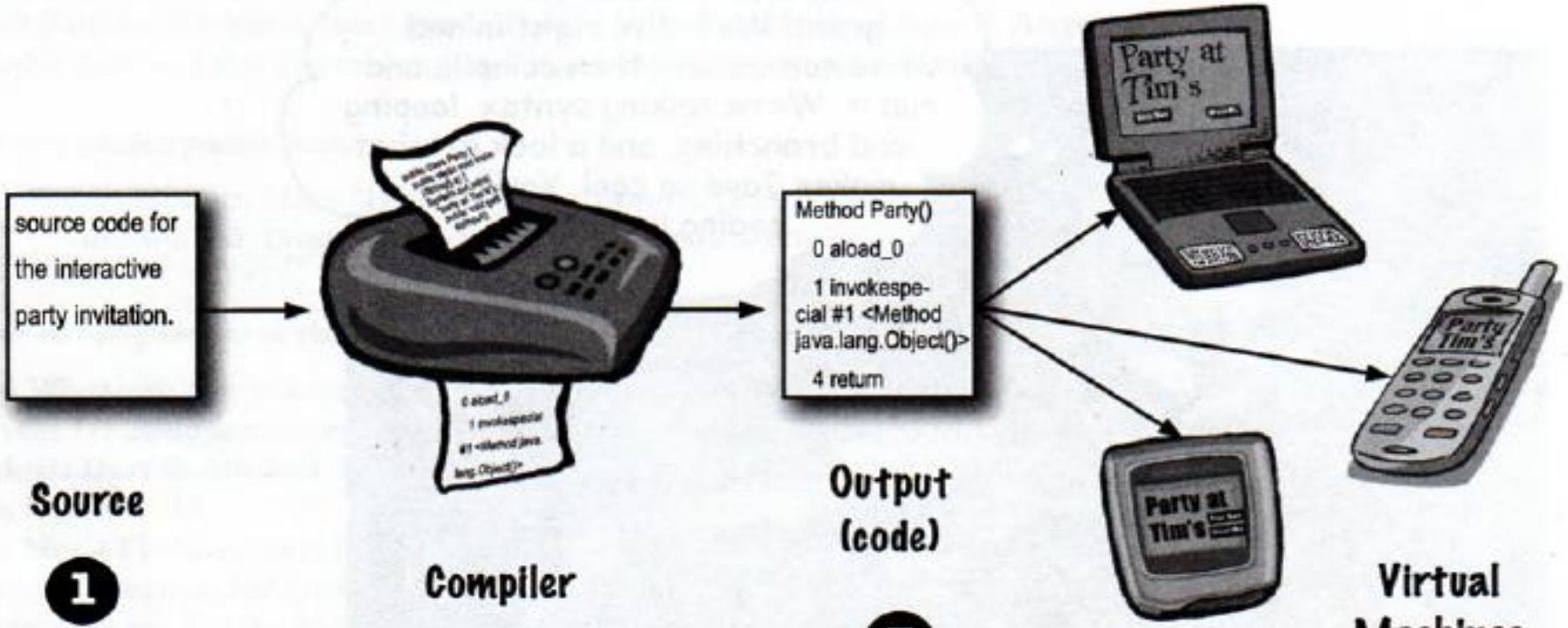
How Java differs from C

- *Java does not include the C unique statement keywords **sizeof** and **typedef**.*
- *Java does not contain the data type **struct** and **union**.*
- *Java does not define the type modifiers keywords **auto**, **extern**, **register**, **signed** and **unsigned**.*
- *Java does not support any explicit pointer type.*
- *Java does not have a preprocessor and therefore we cannot use **#define**, **#include** statements.*
- *Java declares function with no arguments with empty parenthesis and not with **void** keywords as done in C.*
- *Java adds new operators such a **instanceof**.*
- *Java adds labelled **break** and **continue** statements.*

How Java differs from C++

- *Java is true object-oriented language while C++ is basically C with object-oriented extension.*
- *Java does not support operator overloading.*
- *Java does not have template classes as in C++.*
- *Java does not support multiple inheritance of classes. This is accomplished using a new feature called **interface**.*
- *Java does not support global variables. Every variable and method is declared within a class and forms part of that class.*
- *Java does not use pointers and header files.*
- *Java has replaced the destructor function with a garbage collector.*

The way java works:



Source

1

Create a source document. Use an established protocol (in this case, the Java language).

Compiler

2

Run your document through a source code compiler. The compiler checks for errors and won't let you compile until it's satisfied that everything will run correctly.

Output (code)

3

The compiler creates a new document, coded into Java ***bytecode***. Any device capable of running Java will be able to interpret/translate this file into something it can run. The compiled bytecode is platform-independent.

Virtual Machines

4

Your friends don't have a physical Java Machine, but they all have a ***virtual*** Java machine (implemented in software) running inside their electronic gadgets. The virtual machine reads and *runs* the bytecode.

Features of Java

There is given many features of java. They are also known as java buzzwords.

1) Simple
1) Object-Oriented
2) Platform independent
3) Secured
4) Robust
5) Architecture neutral
6) Portable
7) Dynamic
8) Interpreted
9) High Performance
10) Multithreaded
11) Distributed

Simple

According to Sun, Java language is simple because: syntax is based on C++ (so easier for programmers to learn it after C++). removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc. No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

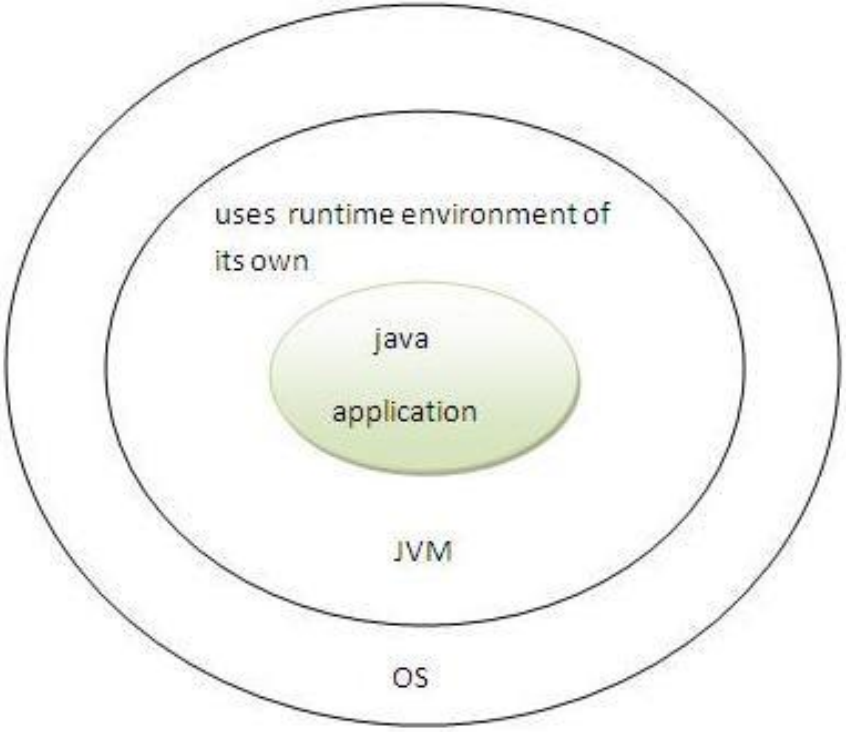
Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior. Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules. Basic concepts of OOPs are:
Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation

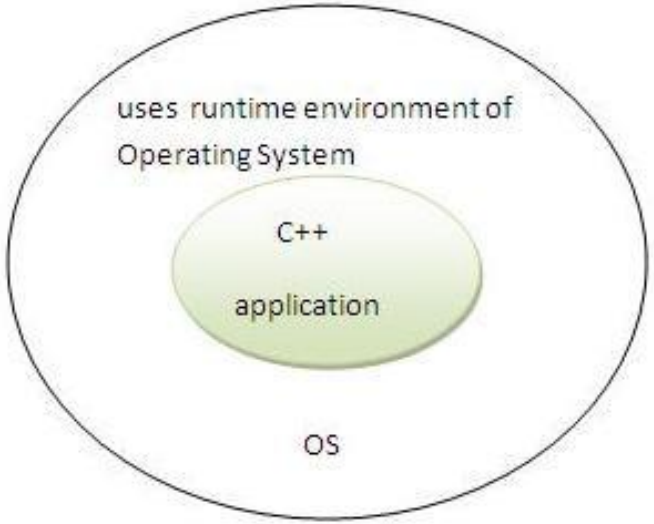
Platform Independent

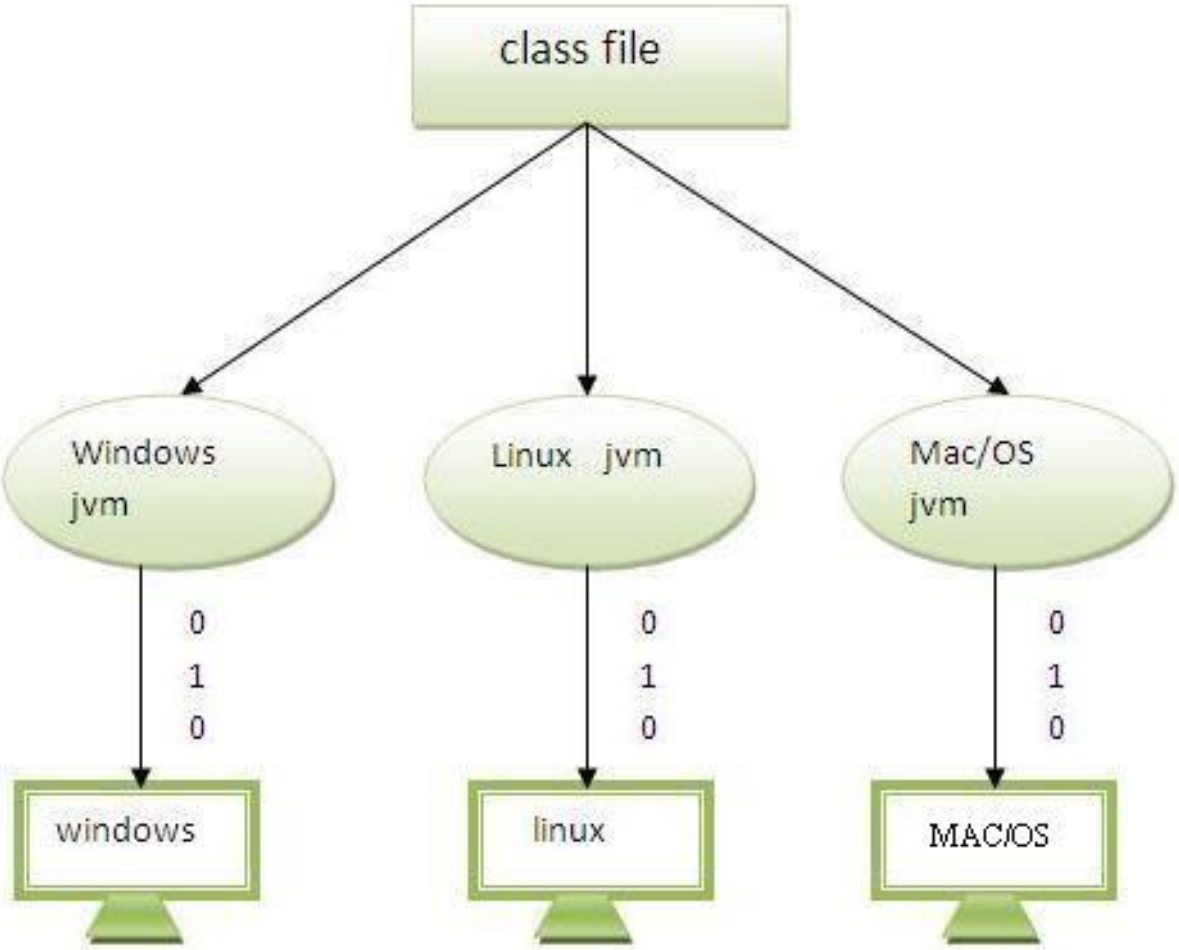
A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.It has two components: Runtime Environment

API(Application Programming Interface)

Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).


uses runtime environment of
its own
java
application
JVM
OS


uses runtime environment of
Operating System
C++
application
OS


class file
Windows
jvm
Linux jvm
Mac/OS
jvm
0
1
0
0
1
0
0
1
0
windows
linux
MAC/OS

Secured

Java is secured because: No explicit pointer

Programs run inside virtual machine sandbox.

Classloader- adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier- checks the code fragments for illegal code that can violate access right to objects.

Security Manager- determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc.

Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is set.

Portable

We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

Basic Concepts of OOP

- *Basic concepts of OOP are :*

1. *Data Abstraction*
2. *Encapsulation*
3. *Inheritance*
4. *Polymorphism*
5. *Dynamic Binding*

Data Abstraction

- ✓ *Abstraction is the process of recognizing and focusing on important characteristics of a situation or object and leaving/filtering out the un-wanted characteristics of that situation or object. Lets take a **person** as example and see how that person is abstracted in various situations*

- ❑ *A doctor sees (abstracts) the **person** as patient.*

  name, height, weight, age, blood group, previous or existing diseases etc

- ❑ *An employer sees (abstracts) a **person** as Employee.*

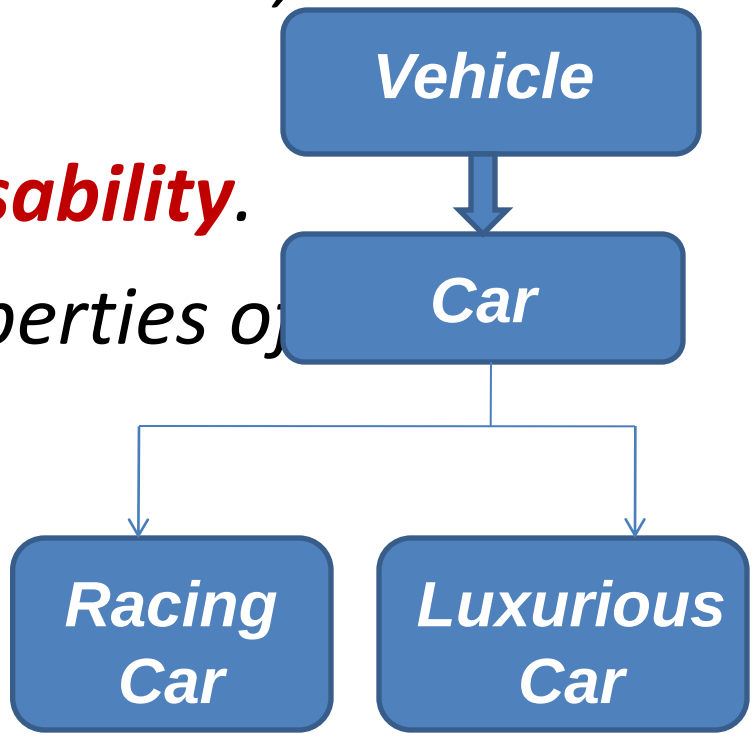
  name, age, health, degree of study, work experience etc

Encapsulation

- ✓Wrapping up to data and objects in a single unit (class) is called encapsulation.
- ✓The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.
- ✓These methods provide the interface between the object's data and the program.
- ✓This insulation of the data from direct access by the program us called *data hiding*.
- ✓In Encapsulation data only performs a specific task and is not concerned about the internal implementation.

Inheritance

- ✓ *Inheritance is the process by which objects of one class acquire the properties of objects of another class.*
- ✓ *Here we have base class and subclass, which have parent-child relationship.*
- ✓ *It provides the concept of **reusability**.*
- ✓ *A subclass inherits all the properties of base class.*
- ✓ *In addition to this, it can add its own properties and behavior.*



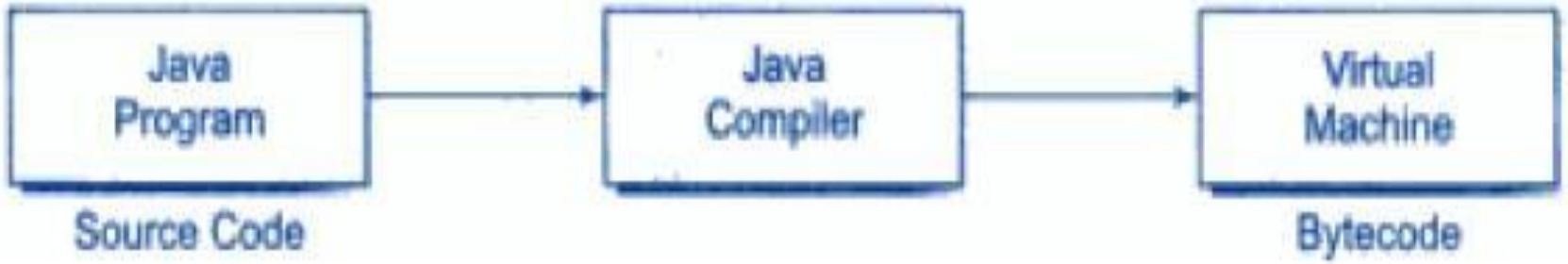
Polymorphism

- ✓ *Polymorphism means the ability to take more than one forms.*
- ✓ *An operation may exhibit different behavior in different instances.*
- ✓ *Eg: Consider operation of addition. For 2 numbers the operation will generate a sum. If the operands are string, then the operation would produce a third string by concatenation.*

Polymorphism cont…

- *Polymorphism is divided into 2 types:*
  1. *Compile time polymorphism – It is at compile time that the child class inherits the public data members and methods of the base class.*
  2. *Runtime polymorphism (**Dynamic binding**) – Dynamic binding means the code associated with a given procedure call is not known until the time of call at runtime.*

Process of compilation



JDK

- *Java Development Kit (JDK) comes with a collection of tools that are used for developing and running java based programs.*
- *The JDK is used to compile Java applications and applets.*
- *JDK needs more Disk space as it contains JRE along with various development tools.*
- *It includes JRE, set of API classes, Java compiler, Web start and additional files needed to write Java applets and applications.*

JDK

- *java – This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment.*
- *javac – the compiler, which converts source code into Java bytecode*
- *jar – the archiver, which packages related class libraries into a single JAR file. This tool also helps manage JAR files.*
- *javadoc – the documentation generator, which automatically generates documentation from source code comments*
- *appletviewer – this tool can be used to run and debug Java applets without a web browser.*

JRE

- *JRE is Java Runtime Environment which is required to run Java Applications also. JRE alone does not contains compiler etc for Java Development.*

- *The java programming language adds the portability by converting the source code to byte code version which can be interpreted by the JRE and gets converted to the platform specific executable ones. Thus for different platforms one has corresponding implementation of JRE. But JRE has to meet the specification JVM (Java Virtual Machine) Concept that serves as a link between the Java libraries and the platform specific implementation of JRE. Thus JVM helps in the abstraction of inner implementation from the programmers who make use of libraries for their programs.*

JDK and JRE

- JRE is a subset of JDK.

  - Two steps for a java program ie.,compile and interpret.
  - JDK does compilation but JRE can't.
  - Both JRE and JDK does interpretation.

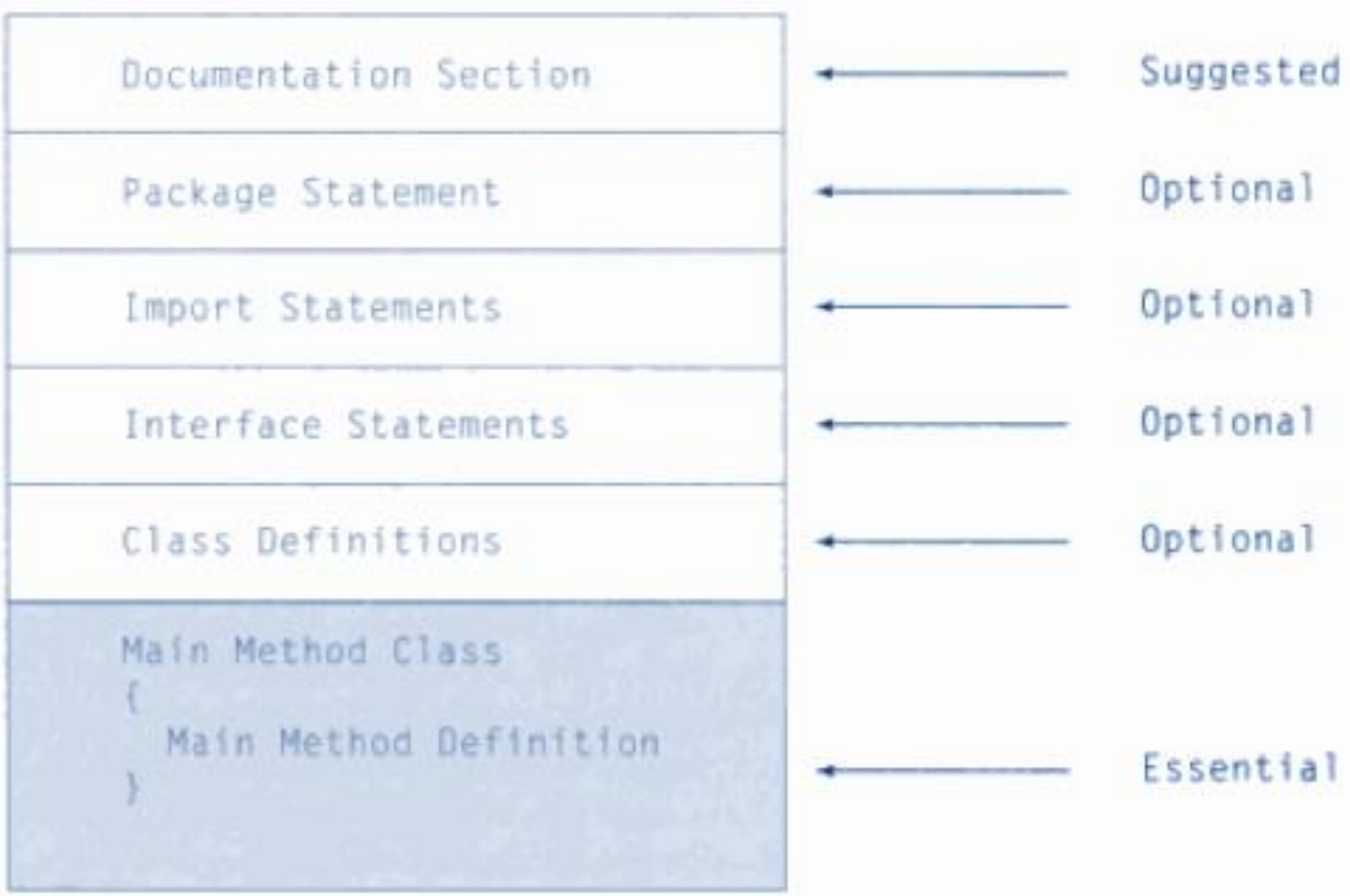
JVM

- *JVM is machine dependent.*
- *The use of the JVM is simple : - To change the byte code into the machine specific code. So the JVM converts the byte code(ie the code which we get when we compile the .java class) into the code that your machine/OS understands.*
- *JVM is machine specific. So JVM for windows will be different from JVM for Mac or Unix.*
- *JVM is one of the most crucial part in the java engine. It is due to JVM only that we say java code is "Compile Once, Run Anywhere" programming language.*

JVM conti…

- *The byte code generated when we compile the code will be THE SAME doesn't matter we compiled the code in Windows OS or some other. However JVM makes sure that the byte code should be interpreted properly in the OS under concerned so interpret the byte code differently for different OS.*

Structure of Java program



Documentation Section:

This section contains set of comments. We can write comment in java within two ways:

1. Line comment - //
2. Block Comment - /* */
3. Documentation Comment is new comment in java - /** */

Package Statement:

This statement declare the package name and inform the compiler that classes declared is contain into this package.

Ex: package student;

Import statement:

This is similar to the #include statement in the c language. It instruct to the interpreter to load the class contained in the given package.

Ex: import student.test;

This statement instruct the interpreter to load the test class contained in the student package.

Interface Statement:

An interface is like a class but include group of methods. Interface is a new concept of java. This is optional and implement when we want to implement multiple inheritance.

Class Definition:

Classes are primary and essential elements of java. Class keyword is used to declare the class. Declare multiple classes in a java program.

Main method class

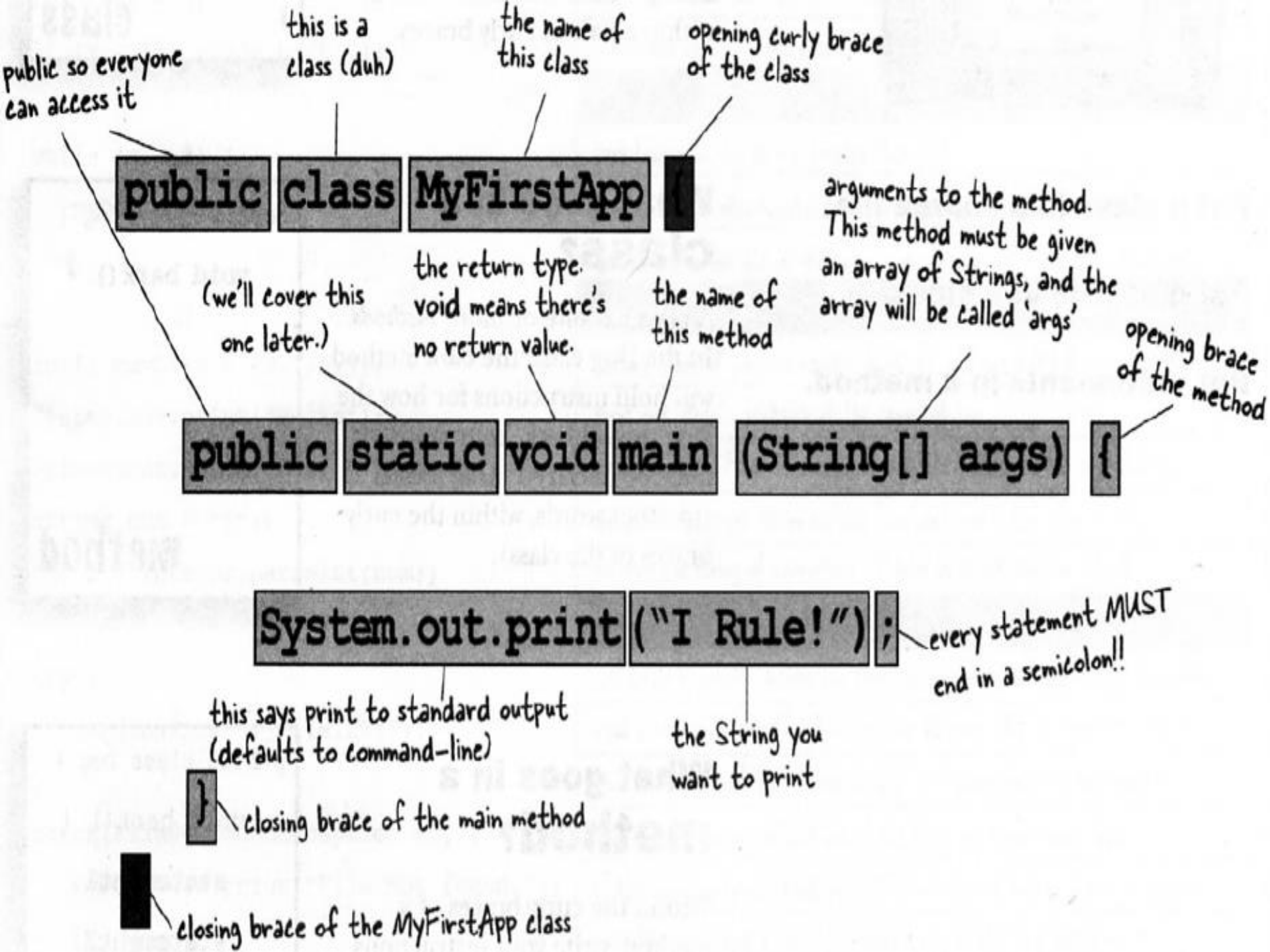
It is essential part of java program. It is starting point of java program. The main method creates objects of various classes and establishes communication between them. On reaching the end of main, the program terminate and the control passes back to the operating system

Example:

```
/* This is sample Java program Develop By: Mr. Bhavesh Patel, Date: 17/7/2014 */

package student

import test.student

class  sample
{
        public static void main(String args[])
        {
                System.out.println(“Hello world”);
        }
}
```

Our first application

```
/** Comment
 * Displays "Hello World!" to the standard output.
 */
public class HelloWorld {

  public static void main (String[] args) {

      //Displays the enclosed String on the Screen Console

      System.out.println("Hello World!");
  }

}
```


public so everyone can access it
this is a class (duh)
the name of this class
opening curly brace of the class
public class MyFirstApp {
(we'll cover this one later.)
the return type. void means there's no return value.
the name of this method
arguments to the method. This method must be given an array of Strings, and the array will be called 'args'
opening brace of the method
public static void main (String[] args) {
System.out.print ("I Rule!") ;
every statement MUST end in a semicolon!!
this says print to standard output (defaults to command-line)
the String you want to print
}
closing brace of the main method
closing brace of the MyFirstApp class

- ✓ *To compile Java code, we need to use the 'javac' tool. From a command line, the command to compile this program is:*

  javac HelloWorld.java

- ✓ *If the compilation is successful, javac will quietly end and return you to a command prompt.*
- ✓ *If you look in the directory, there will now be a HelloWorld.class file. This file is the compiled version of your program. Once your program is in this form, its ready to run.*
- ✓ *To run the program, you just run it with the java command:*

  java HelloWorld

- ✓ *It is important to use the full name with extension when compiling (javac HelloWorld.java) but only the class name when running (java HelloWorld).*

- *When JVM starts running, it searches for the class that you give in the command line. JVM runs everything between { } of your main method.*

- *Every Java application must have at least **one** class and **one** main method (not one main per class, but one main per application).*

1. ***public static void main (String[] args)*** *- defines a method named* ***main****. Every java application must include a main method as interpreter starts its execution at this point.*
2. ***public*** *: This keyword is a access specifier. that declares the main method as unprotected and therefore it is accessible to all other classes.*
3. ***Static*** *: static is the next keyword after public in main method. It declares this method as one that belongs to the entire class and not part of any object of the class. The main method is always declared as static as interpreter uses this method before any objects are created.*
4. **Void**: void is a type modifier that states the main method does not return any value (but simply prints some text on the screen).

Java Comments

- The Java programming language supports three kinds of comments:

1. /* text */
   The compiler ignores everything from /* to */.
2. /** documentation */
   This indicates a documentation comment (doc comment, for short). The compiler ignores this kind of comment, just like it ignores comments that use /* and */. The JDK javadoc tool uses doc comments when preparing automatically generated documentation.
3. // text
   The compiler ignores everything from // to the end of the line.

- System.out.println("Hello World!") – defines JVM to print a message to standard output (default is command-line).
- The **println** method is a member of the out object.
- Out is a static data member of system class.
- This line prints the string

  Hello World!
- The println always appends a newline character to the end of the string.

Scanner Class:

This scanner class is used to read the data into the variable.
This class support jdk1.5 and higher version

Syntax:

Scanner sc = new Scanner (System.in)

Through this code user can get value from the keyboard.

Scanner class comes under java.util.* package so we have to import this class from the util package.

Methods of Scanner class:

nextByte() - read byte data method is
nextLong() - read Long data method is
nextFloat() - read Float data method is
nextShort() - read short data method is
next() - Reading a string, but included white space characters
nextInt() - Reading an integer data.

Program of scanner class:

```
import java.io.*;
import java.util.*;
// scanner class comes under java.util.* package
class scanner
{
        public static void main(String args[])
        {
                Scanner sc = new Scanner(System.in);
                 System.out.println("\n Enter an integer");
                 int x = sc.nextInt();
                 System.out.println("Integer is := " +x);

                 System.out.println("\n Enter a Double");
                 double d = sc.nextDouble();
                 System.out.println("Double is := " +d);

                 System.out.println("\n Enter your name here:");
                 String str = sc.next();
                 System.out.println("Welcome := " +str);
          }

}
```

DataInputStream class:

This class is defined in the package java.io so we have to import it from this class.

This class support various methods to read the data from the console.

This class might throws an exception therefore we have placed the code into the try and catch block.

Syntax:

DataInputStream input = new Data InputStream (System.in)

readLine() – it is used to read string from the keyboard.

The entered value can directly be converted to an integer without storing it to the string variable as

Integer. parseInt(input.readLine())

To convert in Long type following parser is use:
Long.pareInt (input.readLine)

Example of DataInputStream Class:

```
import java.io.*;
class ReadData
{
    public static void main(String args[])
            {
               String  sname;
               //int num;
               try
               {
                                    // Read String Data
                  DataInputStream input = new DataInputStream(System.in);
          System.out.println("Enter the Name:");
                           sname = input.readLine();
                                    System.out.println("Hello" +sname);

                                    // Read Integer Data
                                    String snum;
                                    int num;
                                    System.out.println ("\nEneter the Integer number: ");
                                    snum = input.readLine();
                                    num = Integer.parseInt(snum);
                                    System.out.println("You have entered Integer: " + num);
```

```
// Read Float Value
                                String snum1;
                                Float obj1;
                                float num1;
                                System.out.println ("\nEneter the Float number: ");
                                snum1 = input.readLine();
                                obj1 = Float.valueOf(snum);
                                num1 = obj1.floatValue();
                                System.out.println("You have entered Float Value: " + num1);

                                // Read long Value
                                String snum2;
                                long num2;
                                System.out.println ("\nEneter the Long number: ");
                                snum2 = input.readLine();
                                num2 = Long.parseLong(snum2);

                        System.out.println("You have entered Long Value is: " + num2);
                    }
                    catch (Exception eobj)
                    {
                       System.out.println("Error");
                    }
                }
}
```

Java Tokens

- The smallest individual units in a program are known as **tokens**.
- Java language includes five types of tokens. They are:
  - Reserved keywords
  - Identifiers
  - Literals
  - Operators
  - Seperators

- **Keywords** are the reserved words that have some specific meaning. Java has 50 reserved words.
- **Literals** in java are a sequence of characters (digits, letters and other characters) that represent constant values to be stored in variables. Java has 5 major types of literals:
  1. Integer literals, floating_point literals
  2. Character literals
  3. String literals
  4. Boolean literals
- **Seperators** are symbols to indicate where groups of code are divided and arranged. They basically define the shape and function of our code.

  Eg: Parantheses ( ), braces { }, brackets [ ], semicolon ; , comma, , period .

- **Identifiers** are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java indentifiers follow the following rule:
  1. They can have alphabets, digits and the underscore and dollar sign characters.
  2. They must not begin with a digit.
  3. Upper case and lower case letters are distinct.
  4. They can be of any length.

Operators

- An **Operator** is a symbol that takes one or more arguments and operates on them to produce a result.
  1. Arithmetic operators
  2. Relational operators
  3. Logical operators
  4. Assignment operators
  5. Increment/Decrement operators
  6. Conditional operators
  7. Bitwise operators
  8. Special operators (instanceof, Dot operator)

Instanceof Operator

- The ***instanceof*** is an object reference operator and returns ***true*** if the object on the left-hand side is an instance of the class given on the right hand side. This operator allows us to determine whether the object belongs to a particular class or not.

  Eg: person instanceof student

  It is true if the object person belongs to the class student; otherwise it is false.

- The ***Dot*** operator(.) is used to access the instance variables and methods of class objects.

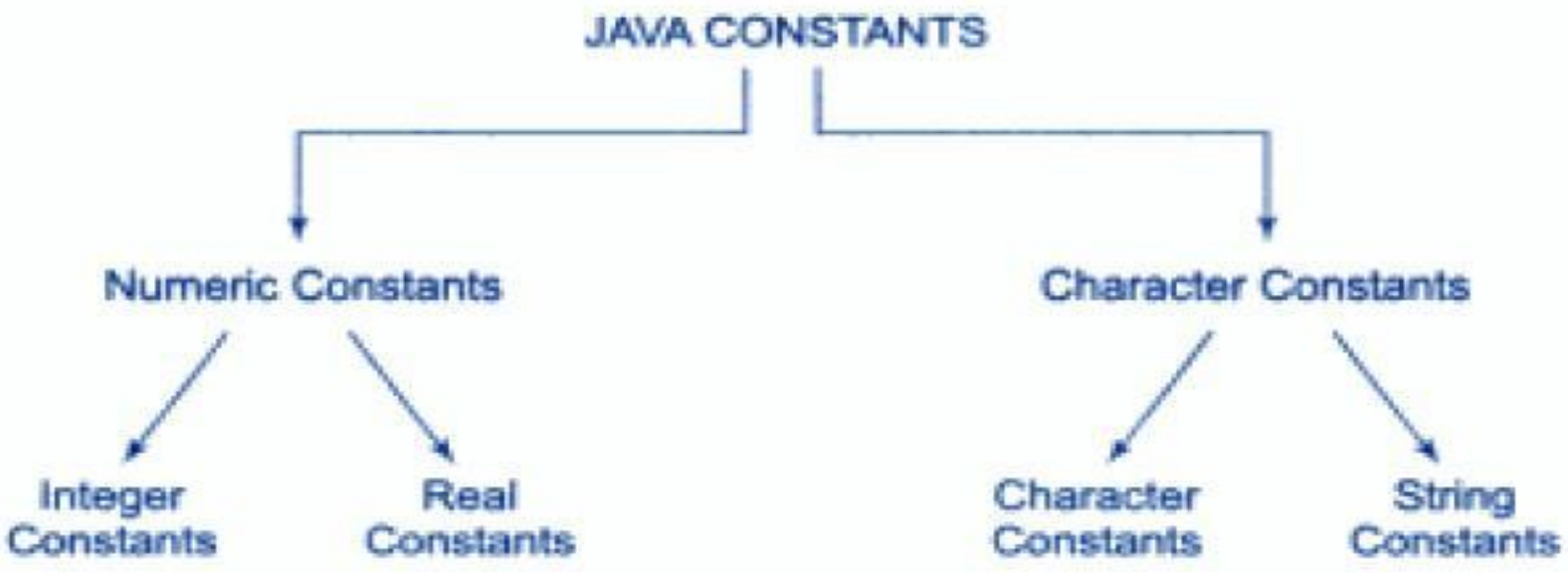
  Eg: person.age

  person.salary()

  It is also use to access classes and sub-packages from a packages.

Constants

- **Constants** are the fixed value that do not change during the execution of the program.

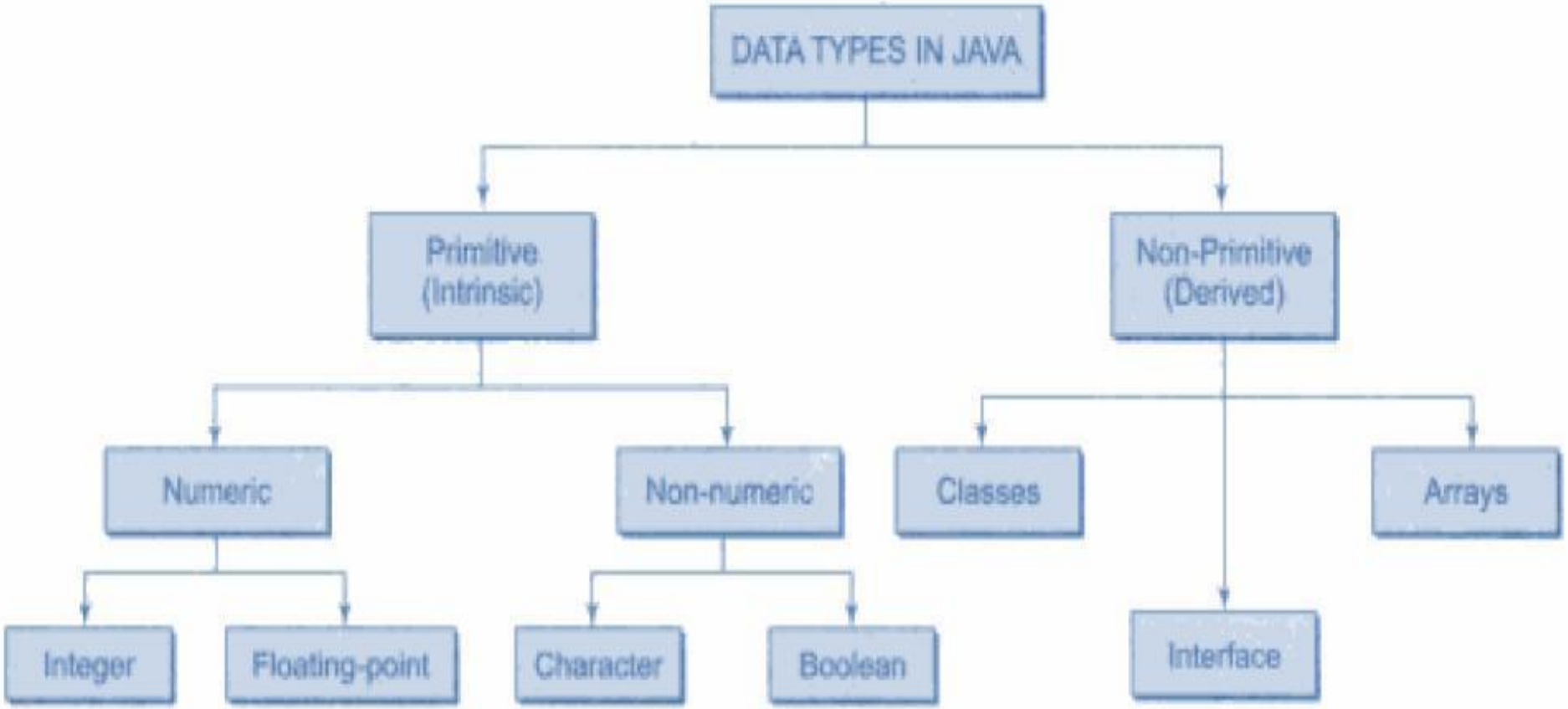


Variables

- **Variable** is a identifier that denotes storage location used to store a data value. A variable may take different values at different times during the execution of the program.
- Rules of declare a variable:
  1. They can have alphabets, digits and the underscore and dollar sign characters.
  2. They must not begin with a digit.
  3. Upper case and lower case letters are distinct.
  4. It should not be a keyword.
  5. White space is not allowed.

Datatypes

- Datatypes specify the size and value that can



Declaration of variable

- **Declaration of variable** The general form of declaration statement is : dataype variable1, variable2;

- Eg: int count;
  float finalPercentage;
  double pi;

Giving values to variables

- A variable must be given value after it is declared. This is achieved in 2 ways:

1. By using assignment statement : variable = value;

   Eg: count = 0;

   finalPercentage = 78.4;

2. By using a read statement : By using readLine() method. This invokes a object of class ***DataInputStream.*** This method reads input from the keyboard as a string which is then converted to corresponding datatype using a wrapper class.

Scope of Variables

- The area of the program where the variable is usable is called ***scope***. Java variables are divided into 3 types:
  1. *Instance* variables
  2. *Class* variables
  3. *Local* variables
- Instance and class variables are declared inside a class.
- ***Class*** variables are global to class and belongs to entire set of objects that class creates. Only one memory location is created for each class variable. They are always static.
- ***Instance*** variables belongs to objects as they are created when the objects are initialized. They take different value for each object.
- Variables declared and used inside a method are called ***local*** variables

Symbolic constants

- Symbolic constants are the constants that can be used throughout the class.
- They should be declared only as a class data member in the beginning of the class.
- They are written in CAPITALS to distinguish from the normal variables.

Eg: final int PASS_MARK = 50;

final int RATE_OF_INTEREST = 8.5;

final float PI = 3.14;

Type casting

- The process of converting one data type to another is called ***casting***.
- The process of assigning a smaller type to a larger one is known as ***widening*** or ***promotion*** *and assigning larger type to smaller one is known as* ***narrowing.***
- Automatic conversion occurs when the destination type has enough precession to store the source value.
  - Eg: int has enough space to hold byte value.

    byte b = 75:

Generic type conversion

- Generics is the enhancement that is given in Java 5.0 version.
- Generics eliminates the need of explicit type casting in collections.
- A collection is a set of interfaces and classes that sort and manipulate a group of data into a single unit.
- Each element is considered as object in collections. Thus we need to typecast it while retrieving.
- Generics determines typecast errors at compile time rather than run time.
- With generics we can specify the type information of data using a parameter.

```
class SampleGenericClass <T> { }
```

Branching Statements (Break and Continue)

Java provides three branching statements break, continue and return.

The break and continue in Java are two essential keyword beginners needs to familiar while using loops (for loop, while loop and do while loop).

break statement in java is used to break the loop and transfers control to the line immediate outside of loop.

continue is used to escape current execution (iteration) and transfers control back to start of the loop.

continue and break statement can be unlabeled or labeled.

Although it's far more common to use break and continue unlabeled.
Let’s understand unlabeled and Labelled continue and break statement using java program.

Below program will do addition of all even numbers of array till it encounter 0 or negative number from an array.

```
public class BreakContinueDemo
{
        public static void main(String[] args)
        {
                int [] numbers = {10,23,19,34,54,23,76,39,65,24,8,0,12,55};
                int sum =0;
               for(int i=0; i<numbers.length; i++)
                {
                        if(numbers[i]<=0)
                        {
                                break;
                        }
                        else if (numbers[i]%2 != 0)
                        {
                                System.out.println(“ ignore odd number );
                                continue;
                        }
                        else
                              sum = sum + numbers[i];
                }
         System.out.println("Sum of all even numbers is = " + sum);
        }
}
```

Labelled break and continue statement:

A label statement must be placed just before the statement being labeled, and it consists of a valid identifier that ends with a colon (:).

You need to understand the difference between labeled and unlabeled break and continue.

The labeled varieties are needed only in situations where you have **nested loop,** and need to indicate which of the nested loops you want to break from, or from which of the nested loops you want to **continue** with the **next iteration.**

```
public class LabledBreakContinueDemo
{
        public static void main(String[] args)
    {
            int breaklimit = 9;
            outer:   // Labelled for Break and Continue Statement
            for (int i = 0; ; i++)
            {
                for (int j = 0; j < 10; j++)
                 {
                        if (j > i)
                         {
                                System.out.println();
                                continue outer;
                        }
                        System.out.print(" " + (i * j));
                }
                if(i==breaklimit)
                {
                        break outer;
                }
            }
  System.out.println();
     }
}
```

Precedence of Java Operators:

Operator precedence determines the grouping of terms in an expression.

This affects how an expression is evaluated.
Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Expression:

Numbers, symbols and operators (such as + and ×) grouped together that show the value of something. Example: 2×3 is an *expression* ...

Example: y = a + b*c / z

Category	Operator	Associativity
Postfix	**() [] . (dot operator)**	**Left to right**
Unary	**++ - - ! ~**	**Right to left**
Multiplicative	*** / %**	**Left to right**
Additive	**+ -**	**Left to right**
Shift	**>> >>> <<**	**Left to right**
Relational	**> >= < <=**	**Left to right**
Equality	**== !=**	**Left to right**
Bitwise AND	**&**	**Left to right**
Bitwise XOR	**^**	**Left to right**
Bitwise OR	**|**	**Left to right**
Logical AND	**&&**	**Left to right**
Logical OR	**||**	**Left to right**
Conditional	**?:**	**Right to left**
Assignment	**= += -= *= /= %= >>= <<= &= ^= |=**	**Right to left**
Comma	**,**	**Left to right**

Class and Object

A class is a user defined data type.

A *class* is a collection of *fields* (data) and *methods* (procedure or function) that operate on that data.

The basic syntax for a class definition:

```
class  ClassName [extends SuperClassName]
{
        [fields declaration]
        [methods declaration]
}
```

Example:

```
public class Circle
{
    // my circle class
}
```

Adding Fields: Class Circle with fields

Add *fields*

```
public class Circle
{
        public double x, y;  // centre coordinate
        public double r;     //  radius of the circle
}
```

The fields (data) are also called the ***instance*** varaibles.

Adding Methods

A class with only data fields has no life. Objects created by such a class cannot respond to any messages.

Methods are declared inside the body of the class but immediately after the declaration of data fields.

The general form of a method declaration is:

```
type MethodName (parameter-list)
{
        Method-body;
}
```

Adding Methods to Class Circle

```
public class Circle {

    public double x, y; // centre of the circle
    public double r;    // radius of circle

    //Methods to return circumference and area
    public double circumference() {
                return 2*3.14*r;
    }
    public double area() {
                return 3.14 * r * r;
    }
}
```

Object:

An Object in java is essentially a block of memory that contains space to store all the instance variables.

Creating an object is also referred to as **instantiating** an Object

Object in java is created using **new** operator.

The new operator creates an object of the specified class and returns a reference to that object.

Example:
Rectangle rect1 ; // Declare an object
Rect1 = new Rectangle ( ) ; // instantiate the object

Both statements can be combined into one as shown below:

Rectangle rect1 = new Rectangle ();
Rectangle rect2 = new Rectangle ();

We can create **any number** of objects of the class.

ACCESSING CLASS MEMBERS:

Each object contains class variables and methods before using this we should assign values to these variables in order to use them in our program.

Here . (dot) operator is used to access class variables and methods with an object.

Syntax:

Objectname.varibalename = vaue ;
Objectname.methodname(parameter list) ;

Example:

Rectangle rect1 = new Rectangle ();
Rectangle rect2 = new Rectangle ();

Rect1. length = 15;
Rect1.width = 10 ;

Rect2.length = 11;
Rect2.width = 12;

Example of class and object:

```
Class Rectangle
{
        int length, width;
        void getdata (int x, int y)
        {
                length = x;
                width = y
        }

        int rectarea()
        {
                int area = length * width ;
                return (area);
        }

}
```

```
Class RectArea
{
        public static void main(String args[])
        {
                int area1, area2;
                Rectangle rect1 = new Rectangle ();
                Rectangle rect2 = new Rectangle ();

                rect1.length = 15;
                rect1.width = 10;
                area1 = rect1.length * rect1.width;

                rect2.getdata(20,12)
                area2 = rect2.rectarea();

                System.out.println (“Area of Rect1 is:” + area1);

                System.out.println (“Area of Rect2 is:” + area2);

        }
}
```

java Constructor:

Java constructors are the methods which are used to initialize objects.

Constructor method has the same name as that of class.

Constructors are called or invoked when an object of class is created and can't be called explicitly.

Attributes of an object may be available when creating objects if no attribute is available then **default constructor** is called.

Constructor without parameters are called **default constructor.**

Constructor with parameters are called **parameterized constructor.**

```
class Programming
 {
                    //constructor method
          Programming()   // Default constructor
          {
                    System.out.println("Constructor method called.");
          }

          public static void main(String[] args)
          {
                    Programming object = new Programming(); //creating object
          }
}
```

Java constructor overloading and parameterized constructor:

java constructor can be overloaded

i.e. we can create as many constructors in our class as desired.

Number of constructors depends on the information about attributes or parameters of an object we have while creating objects.

Which constructor will be called depends upon the no. of parameters passed in the object

See constructor overloading example:

```
class Language
{
        String name;
        Language()
        {
                System.out.println("Constructor method called.");
        }

        Language(String t)    // Parameterized constructor
         {
                name = t;
         }

         void setName(String t)
        {
                name = t;
         }

        void getName()
        {
                System.out.println("Language name: " + name);
        }
```

```
	public static void main(String[] args)
	{
		Language cpp = new Language();
		Language java = new Language("Java");

		cpp.setName("C++");
		java.getName();
		cpp.getName();
	}

}
```

Constructor example for cuboid:

```
public class Cube1
{
        int length ;
        int breadth ;
        int height ;

        public int getVolume( )
        {
                return ( length * breadth * height );
        }
        Cube1()
        {
                length = 10;
                breadth = 10;
                height = 10;
        }

        Cube1(int l, int b, int h)
        {
                length = l;
                breadth = b;
                height = h;
        }
```

```
    public static void main(String[] args)
    {
        Cube1 cubeObj1, cubeObj2;

        cubeObj1 = new Cube1();
        cubeObj2 = new Cube1(10, 20, 30);


        System.out.println("Volume of Cube1 is :
"+cubeObj1.getVolume());
        System.out.println("Volume of Cube2 is :
"+cubeObj2.getVolume());
    }
}
```

Destructor and Garbase collection:

The **finalize()** method is equivalent to a **destructor** of C++.

When the job of an object is over, or to say, the object is no more used in the program, the object is known as **garbage**.

The process of removing the object from a running program is known as **garbage collection**.

Garbage collection frees the memory and this memory can be used by other programs or the same program further in its execution.

Before an object is garbage collected, the JRE (Java Runtime Environment) calls the finalize() method. finalize() method can be best utilized by the programmer to close the I/O streams.

Syntax:

protected void finalize( ) throws Throwable

```
public class Demo
{
        static Demo d1, d2 ;
        public void show( )
        {
                System.out.println("Hello 1");
        }

        protected void finalize( ) throws Throwable
        {
            if(d1 != null)
            {
                System.out.println("d1 object is not eligible for garbage collection and is still active");
                d1 = null;
                if (d1 == null)
                System.out.println("d1 is not referenced and getting removed from memory");
            }

           if(d2 != null)
           {
                System.out.println("d2 object is not eligible for garbage collection and is still active");
                d2 = null;
                if(d2 == null)
                        System.out.println("d2 is not referenced and getting removed from memory");
            }

          super.finalize( );
      }
```

```
    public static void main( String args[])
    {
            d1 = new Demo( );
            d2 = new Demo( );

            d1.show( );
            d2.show( );

        System.runFinalizersOnExit(true);
    }
}
```

System class comes with the following two methods for garbage collection.

System.runFinalization( ): removes all objects ready of garbage collection.

System.runFinalizersOnExit(true): advice to call finalize() method before garbage collection.

Method Overloading:

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

Advantage of method overloading?
Method overloading **increases the readability of the program**.

Different ways to overload the method
There are two ways to overload the method in java.

1. By changing number of arguments
2. By changing the data type

1)Example of Method Overloading by changing the no. of arguments

```
class Calculation
{
        void sum(int a,int b)
        {
                System.out.println(a+b);
        }

        void sum(int a,int b,int c)
        {
                System.out.println(a+b+c);
        }

          public static void main(String args[])
        {
                Calculation obj=new Calculation();
                obj.sum(10,10,10);
                obj.sum(20,20);

         }
}
```

2)Example of Method Overloading by changing data type of argument

```
class Calculation
{
        void sum(int a,int b)
        {
                System.out.println(a+b);
        }

        void sum(double a, double b)
        {
                System.out.println(a+b);
        }

        public static void main(String args[])
        {
            Calculation obj=new Calculation();
            obj.sum(10.5,10.5);
            obj.sum(20,20);

          }
}
```

Method Overloaing is not possible by changing the return type of method because it creates the ambiguity.

Example:

```
class Calculation
{
        int sum(int a,int b)
        {
                System.out.println(a+b);
        }
        double sum(int a,int b)
        {
                System.out.println(a+b);
        }

  public static void main(String args[])
  {
          Calculation obj=new Calculation();
          int result=obj.sum(20,20); //Compile Time Error

  }
}
```

Note: main () overloadin is possible.

Static keyword:

The **static keyword** is used in java mainly for memory management. We may apply static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

variable (also known as class variable)
method (also known as class method)
block
nested class

1) static variable

If you declare any variable as static, it is known static variable. The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.

The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable
It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

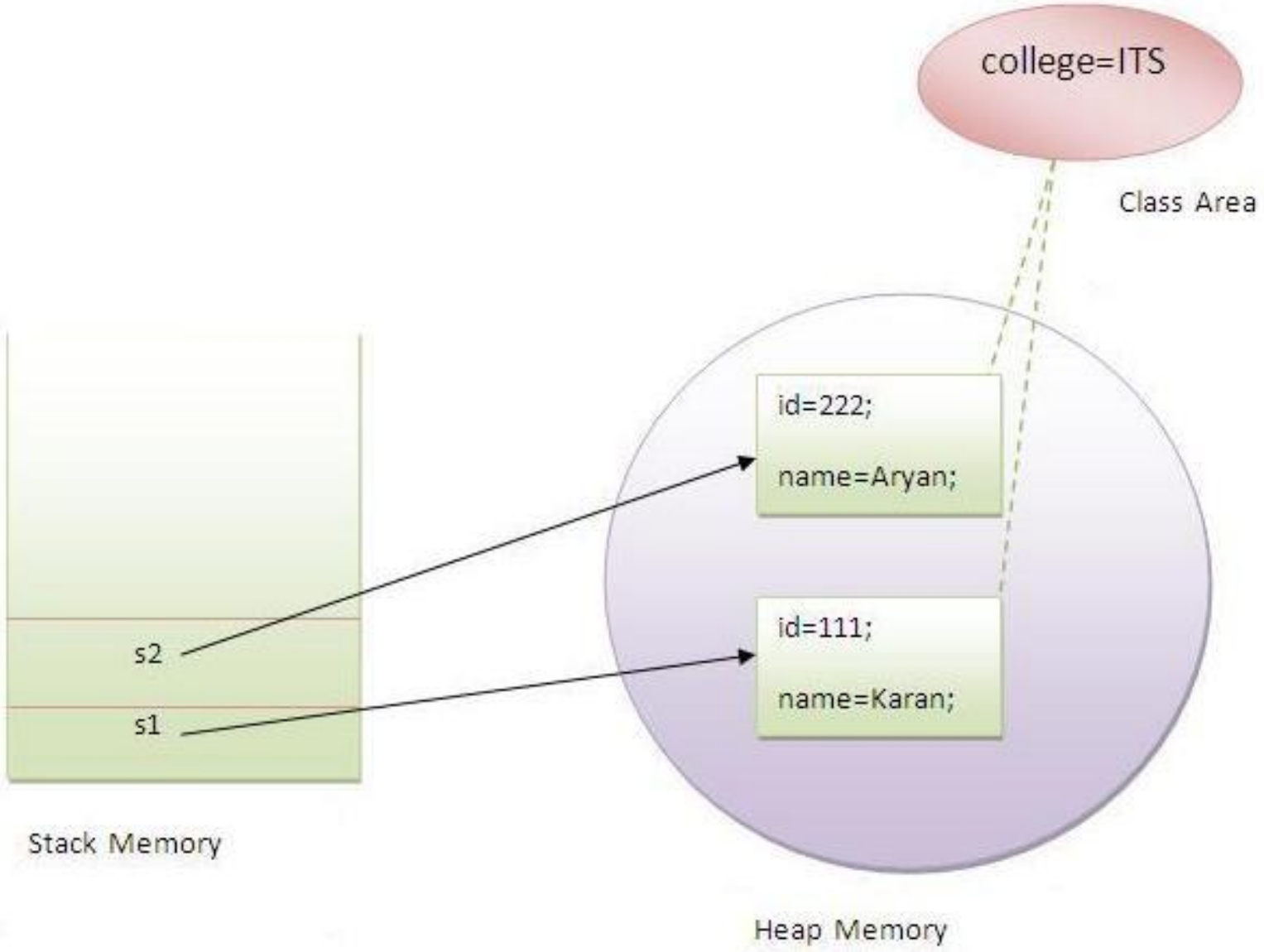
```
class Student
{
    int rollno;
    String name;
    String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.

Note: static property is shared to all objects.

Example of static variable

```
class Student
{
        int rollno;
        String name;
        static String college ="ITS";

    Student(int r,String n)
    {
      rollno = r;
      name = n;
    }
          void display ()
          {
                    System.out.println(rollno+" "+name+" "+college);
          }

    public static void main(String args[])
    {
     Student s1 = new Student (111,"Karan");
     Student s2 = new Student (222,"Aryan");

     s1.display();
     s2.display();
     }
}
```


college=ITS
Class Area
id=222;
name=Aryan;
id=111;
name=Karan;
s2
s1
Stack Memory
Heap Memory

Program of counter without static variable

```
class Counter
{
        int count=0;//will get memory when instance is created

        Counter()
        {
            count++;
            System.out.println(count);
        }
        public static void main(String args[])
        {
            Counter c1=new Counter();
            Counter c2=new Counter();
            Counter c3=new Counter();
        }
}
```

Output: **1**
1
1

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
class Counter
{
        static int count=0;   //will get memory only once and retain its value

        Counter()
        {
            count++;
            System.out.println(count);
        }

    public static void main(String args[])
    {

    Counter c1=new Counter();
    Counter c2=new Counter();
    Counter c3=new Counter();
     }

}
```

Output: 1 2 3

2) static method

If you apply static keyword with any method, it is known as static method

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Example of static method

//Program of changing the common property of all objects(static field).

```
class Student
{
    int rollno;
    String name;
    static String college = "ITS";

    static void change()
    {
            college = "BBDIT";
    }

            Student(int r, String n)
            {
                    rollno = r;
                    name = n;
            }
```

```
        void display ()
        {
                System.out.println(rollno+" "+name+" "+college);
        }

        public static void main(String args[])
        {
                Student.change();

              Student s1 = new Student (111,"Karan");
              Student s2 = new Student (222,"Aryan");
              Student s3 = new Student (333,"Sonoo");

              s1.display();
              s2.display();
              s3.display();
         }
}
```

Output:

111 Karan BBDIT
222 Aryan BBDIT
333 Sonoo BBDIT

Restrictions for static method

There are two main restrictions for the static method.

They are:

- The static method can not use non static data member or call non-static method directly.
- this and super cannot be used in static context.

```
class A
{
 int a=40;//non static

       public static void main(String args[])
       {
         System.out.println(a);
        }
}
```

Output:Compile Time Error

Que) why main method is static?

Ans) because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation.

3)static block

Is used to initialize the static data member.

It is executed before main method at the time of classloading.

Example of static block

```
class A{

  static
   {
       System.out.println("static block is invoked");
   }

     public static void main(String args[])
     {
              System.out.println("Hello main");
     }
}
```

Output:static block is invoked Hello main

Que)Can we execute a program without main() method?

Ans)Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

```
class A
{
  static
  {
        System.out.println("static block is invoked");
        System.exit(0);
  }
}
```

Output: static block is invoked (if not JDK7)

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The final keyword can be used in many context. Final can be:

variable
method
class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable:

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike
{
 final int speedlimit=90;//final variable
 void run()
{
  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike obj=new  Bike();
 obj.run();
 }
}//end of class
```

Output:Compile Time Error

2) final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike
{
  final void run(){System.out.println("running");}
}

class Honda extends Bike
{
   void run()
   {
        System.out.println("running safely with 100kmph");
    }

   public static void main(String args[])
  {
          Honda honda= new Honda();
          honda.run();
   }
}
```

Output:Compile Time Error

3) final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}

class Honda extends Bike
{
  void run()
{
System.out.println("running safely with 100kmph");
}

  public static void main(String args[])
{
  Honda honda= new Honda();
  honda.run();
  }
}
```

Output:Compile Time Error

Is final method inherited?

Yes, final method is inherited but you cannot override it.

For Example:

```
class Bike
{
     final void run()
     {
          System.out.println("running...");
      }
}
class Honda extends Bike
{
   public static void main(String args[])
   {
    new Honda().run();
   }
}
```

Output:running...

Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

Example of blank final variable

```
class Student
{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example of static blank final variable

```
class A
{
  static final int data;//static blank final variable
  static
  {
        data=50;
   }
  public static void main(String args[])
  {
    System.out.println(A.data);
  }
}
```

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

```
class Bike
{
  int cube(final int n)
  {
   n=n+2;//can't be changed as n is final
   n*n*n;
  }
  public static void main(String args[])
  {
    Bike b=new Bike();
    b.cube(5);
  }
}
```

Output:Compile Time Error

Q) Can we declare a constructor final?

No, because constructor is never inherited.

Nested and Inner Classes

- a class within another class; such classes are known as nested classes.
- The scope of a nested class is bounded by the scope of its enclosing class.
- A nested class has access to the members, including private members, of the class in which it is nested. However, the enclosing class does not have access to the members of the nested class.
- There are two types of nested classes: *static* and *non-static*.
- The most important type of nested class is the *inner* class. An inner class is a non-static nested class.

Inheritance in JAVA:

Inheritance is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance is that you can create new classes that are built upon existing classes.

When you inherit from an existing class, you reuse (or inherit) methods and fields, and you add new methods and fields to adapt your new class to new situations.

Inheritance represents the **IS-A relationship**.

Why use Inheritance?

For Method Overriding (So Runtime Polymorphism).

For Code Reusability.

The keyword extends indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a superclass. The new class is called a subclass.

Syntax of Inheritance

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

Example:

```
class Employee
{
                float salary=40000;
}

class Programmer extends Employee
{
                 int bonus=10000;

                 public static void main(String args[])
                {
                        Programmer p=new Programmer();
                        System.out.println("Programmer salary is:"+p.salary);
                        System.out.println("Bonus of Programmer is:"+p.bonus);
                }
}
```

Output:

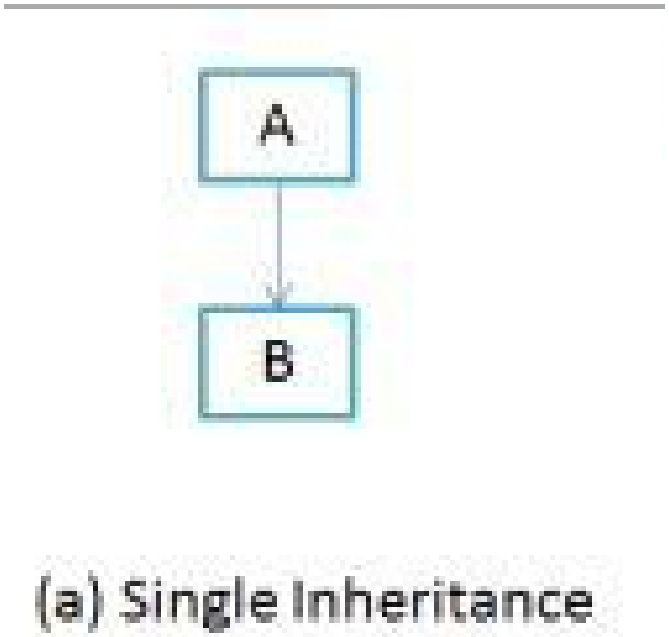
Programmer salary is:40000.0

Bonus of programmer is:10000

Types of Inheritance:

1) Single Inheritance

Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



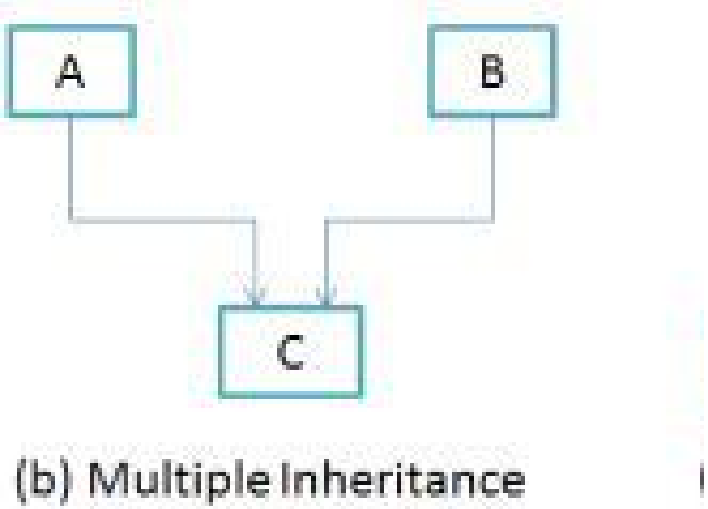
(a) Single Inheritance

Single Inheritance example program in Java

```
Class A
{
        public void methodA()
        {
                  System.out.println("Base class method");
        }
 }

Class B extends A
{
        public void methodB()
         {
                System.out.println("Child class method");
         }

       public static void main(String args[])
       {
                B obj = new B();
                obj.methodA(); //calling super class method obj.methodB(); //calling local method
       }
}
```

2) Multiple Inheritance

"**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



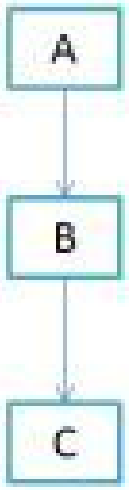
(b) Multiple Inheritance

Note 1: Multiple Inheritance is very rarely used in software projects. Using Multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

Note 2: Most of the new OO languages like **Small Talk, Java, C# do not support Multiple inheritance**. Multiple Inheritance is supported in C++.

3) Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



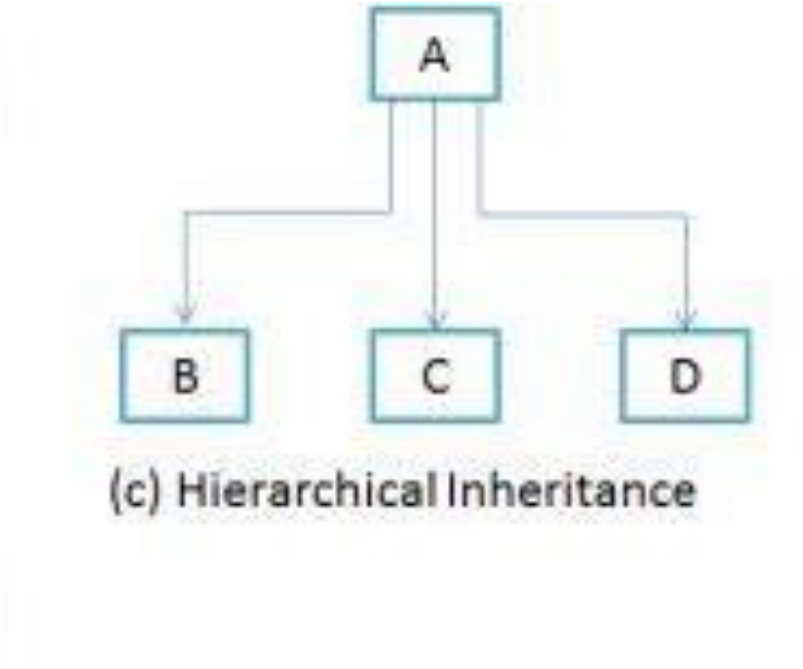
(d) Multilevel Inheritance

Multilevel Inheritance example program in Java

```
Class X
{
      public void methodX()
     {
            System.out.println("Class X method");
     }
}

Class Y extends X
{
       public void methodY()
      {
             System.out.println("class Y method");
      }
}

Class Z extends Y
{
        public void methodZ()
        {
              System.out.println("class Z method");
        }

  public static void main(String args[])
    {
            Z obj = new Z();
            obj.methodX(); //calling grand parent class method obj.methodY(); //calling parent class method
            obj.methodZ(); //calling local method
    }
}
```

4) Hierarchical Inheritance

In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D.



(c) Hierarchical Inheritance

Example of Hierarchical Inheritance:

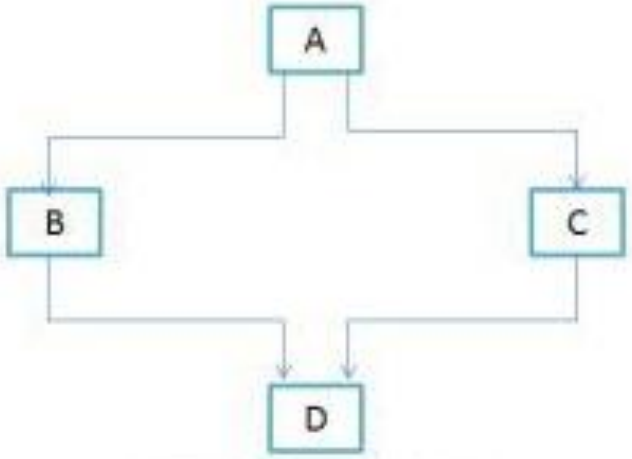
```
Class A
{
        public void methodA()
         {
                System.out.println("method of Class A");
        }
}

Class B extends A
{
        public void methodB()
        {
                System.out.println("method of Class B");
        }
 }

Class C extends A
{
         public void methodC()
        {
                System.out.println("method of Class C");
        }
}

Class D extends A
{
        public void methodD()
        {
                System.out.println("method of Class D");
        }
}
```

```
Class MyClass
{
        public void methodB()
        {
                System.out.println("method of Class B");
        }

        public static void main(String args[])
        {
                B obj1 = new B();
                C obj2 = new C();
                D obj3 = new D();

                obj1.methodA();
                obj2.methodA();
                obj3.methodA();
        }

}
```

Output:
method of Class A
method of Class A
method of Class A

5) Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple inheritance.** A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.



(e) Hybrid Inheritance

Why Hybrid Inheritance not possible in java:

As you can see in the above diagram that it's a combine form of single and multiple inheritance. Since java doesn't support multiple inheritance, the hybrid inheritance is also not possible.

Case 1: Using classes: If in above figure B and C are classes then this inheritance is not allowed as a single class cannot extend more than one class (Class D is extending both B and C). Reason explained below!!

Case 2: Using Interfaces: If B and C are interfaces then the above hybrid inheritance is allowed as a single class can implement any number of interfaces in java.

Use of super keyword in Inheritance:

super keyword is used to explicitly invoke a **superclass constructor.**

If you use this form, it must appear as the first statement of the constructor to ensure that the superclass constructor executes before the subclass constructor in java.

Syntax to call Superclass Constructor

super(args);

args is an optional list of arguments of the superclass constructor.

Example of Super keyword to call Constructor of super class

```
class Person
{
    String FirstName;
    String LastName;

    Person(String fName, String lName)
    {
            FirstName = fName;
            LastName = lName;
     }

}


class Student extends Person
{
    int id;
    String standard;
    String instructor;

    Student(String fName, String lName, int nId, String stnd, String instr)
    {
        super(fName,lName);
        id = nId;
        standard = stnd;
        instructor = instr;
     }

}
```

```
class SuperKeywordForConstructorDemo
{
	public static void main(String args[])
	{
		Student sObj = new Student("Jacob","Smith",1,"1 - B","Roma");

		System.out.println("Student :");
		System.out.println("First Name : " + sObj.FirstName);
		System.out.println("Last Name : " + sObj.LastName);
		System.out.println("ID : " + sObj.id);
		System.out.println("Standard : " + sObj.standard);
		System.out.println("Instructor : " + sObj.instructor);
	}
}
```

Method Override:

Method in the subclass that has the same name, same arguments and same return type as a method in the super class. Then when the method is called, the method defined in the subclass is invoke and executed instead of the one in the super class. **This is known as method overriding.**

In method overriding, if you want to call the super class's method by using the subclass object for that **super keyword** is used.

Example:

```
public class Superclass
{
        public void printMethod()
        {
                System.out.println("Printed in Superclass.");
        }
}
public class Subclass extends Superclass
{
        public void printMethod()
        {
                // super.printMethod();
                System.out.println("Printed in Subclass");
        }

        public static void main(String[] args)
        {
                Subclass s = new Subclass();
                s.printMethod();
        }
}
```

Output:

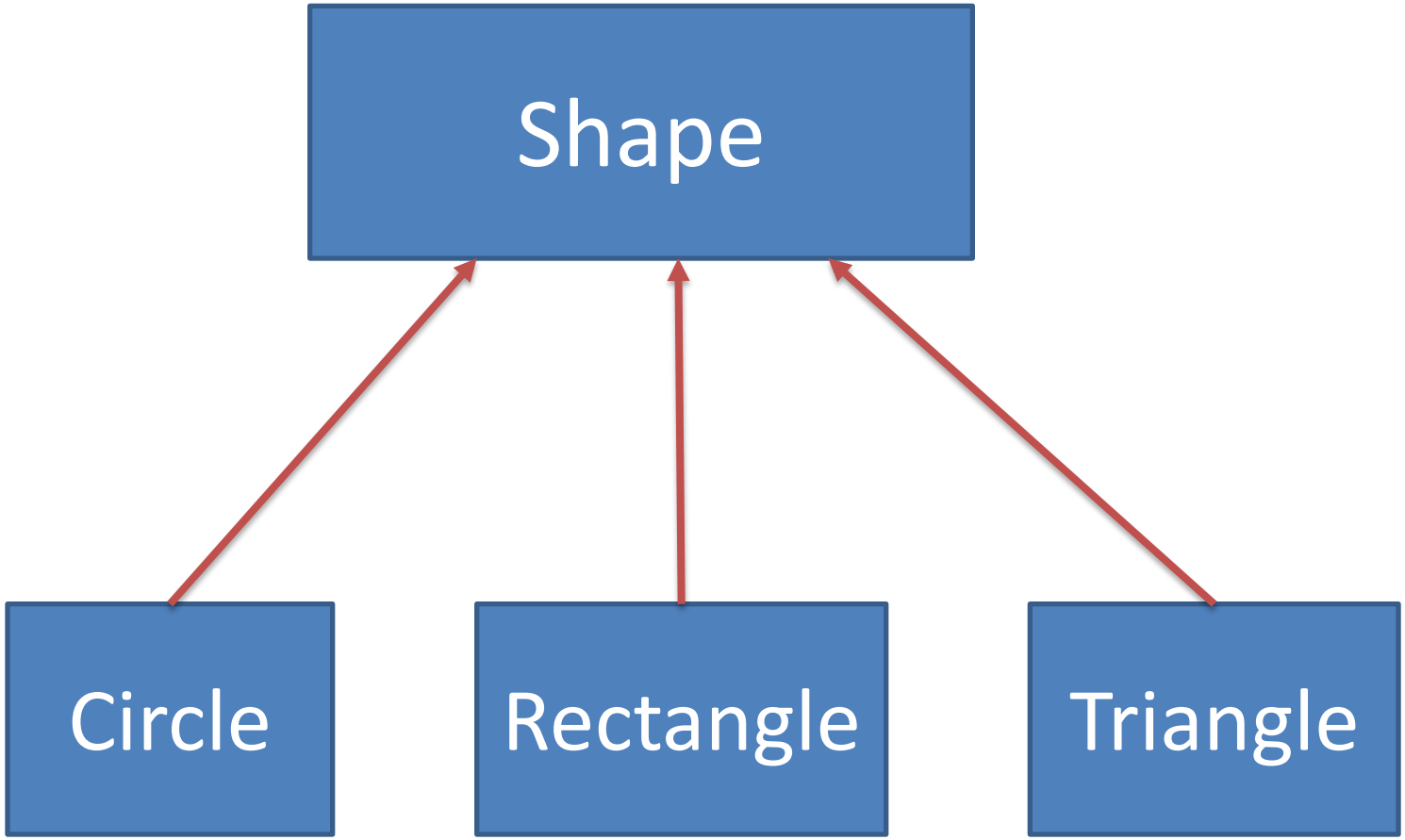
Printed in Superclass.
Printed in Subclass

Abstract keyword:-

- By making a method final we ensure that the method is not redefine in a sub-class i.e. method can never be sub classed.
- Java allows us to do something that is exactly opposite to this. That is we can indicate method must always be redefine in a sub class. Thus, making overriding compulsory.
- This is done using modifier keyword abstract in the method definition.

Example:-

- Consider geometrical shapes such as circle, rectangle and triangle.
- All of them have certain common properties like area.
- the common properties can be abstracted and brought under a common class.
- But this method cannot have body or definition.
- So, method's are define without a body or definition.


Shape
Circle
Rectangle
Triangle

- Java allows declaration of method without body such method's are called abstract method's.
- Abstract methods are define with a keyword abstract. A class containing at least one abstract method is called abstract class.
- Since abstract class do not have concrete methods so object for abstract classes cannot be created.
- When an abstract class is define a sub class of the abstract class must give a concrete method.
- Otherwise the sub class is also become an abstract class.

Example of Abstract Keyword:-

```
abstract class shape
{
        double pi=3.14,area;
        abstract void area();

}
class circle extends shape
{
        int r;
        circle(int radius)
        {
                r=radius;
        }
        void area()
        {
                area=(pi*r*r);
                System.out.println("Area for Circle Is:  "+area);
        }
}
```

```
class rectangle extends shape
{
		int l,b;
		rectangle(int x,int y)
		{
			l=x;
			b=y;
		}
		void area()
		{
			area=2*(l+b);
			System.out.println("Area of rectangel is:  "+area);
		}
}
class demoabst
{
		public static void main(String args[])
		{
			circle ob=new circle(10);
			rectangle re=new rectangle(2,7);
			ob.area();
			re.area();
		}
}
```

this keyword or instance variable hiding:-

In declaring methods and constructors parameters are used as argument the name of parameter are generally different from that the name of instance variable however java allows parameter and instance variable to have the same name. In order to identify instance variables from formal System.out.println parameters the keyword "THIS" is used. **this** refers to the current object.

```
class demothis
{
        int x,y;
        demothis(int x,int y)
        {
                this.x=x;
                this.y=y;
        }
}
class demotest
{
        public static void main(String args[])
        {
                demothis d1=new demothis(20,10);
                System.out.println("The value of x is" +d1.x);
                System.out.println("The vlaue of y is" +d1.y);
        }
}
```

Usage of java this keyword
Here is given the 6 usage of java this keyword.

this can be used to refer current class instance variable.
this can be used to invoke current class method (implicitly)
this() can be used to invoke current class constructor.
this can be passed as an argument in the method call.
this can be passed as argument in the constructor call.
this can be used to return the current class instance from the method.

INTERFACE:-

- JAVA does not supports multiple inheritance that is class F extends A,B,C.
- Interface are similar to abstract class but differ in their functionality. Interface is defines method without body. Interfaces can not have instance variable but it can contain final variable which must be initialize with values the method can have type signature the interface helps to implement multiple inheritance in Java.
- Once an interface is define another class should implement all the methods of interface.
- An interface can be implement in any member of classes.
- A class can implement any number of interfaces.

The general form of implementing an interface is,

class class_name [Extend super class] implements interface_A,...N)

- In the implementing class methods are to be given for the abstract methods define in the interface.

Example of interface:-

```
interface area
{
        final float pi=3.14f;
        public void compute(float x,float y);
}
class rectangle implememts area
{
        public void compute(float x,float y)
        {
                System.out.println("Mulitiplication is" +(x*y));
        }
}
class c implements area
{
        public void compute(float x,float y)
        {
                System.out.println("Multiplication is" +(pi*x*x));
        }
}
class demointer
{
        public Static void main(String args[])
        {
                rectangle r=new rectangle();
                c c1=new c();
                r.compute(2.0f,3.0f);
                c1.compute(2.0f,0.0f);
        }
}
```

Difference between class & interface

CLASS

- The member of a class can be constant or variable.
- The class definition can contain the code for each of its methods i.e. the method can be abstract or non-abstract.
- It can be instantiated by declaring objects.
- It can use various access specifies like public, private or protected.

INTERFACE

- The member of a interface are always declared as constant i.e. there values are final.
- The methods in an interface are abstract in nature.
- It can not be used to declared objects it can only be inherited by a class.
- It can only use the public access specifies.

Difference between abstract and interface:-

Abstract

- Abstract keyword is used to create an abstract class it can be used with method.
- Sub classes use extend keyword to extend an abstract class.
- Abstract classes can have methods with implementation.
- Abstract class can have constructors.
- we can use abstract keyword to make a class abstract. Abstract class methods can have access modifiers as public , private, protected, static.
- Abstract classes can extend other class and implemenet interfaces

Interface

- Interface keyword is used to create interface and it cannot be used with methods.
- Sub class uses implement keyword to implement interface.
- Whereas interface provides absolute abstraction and cannot have any implementation.
- Interface cannot have any constructor.
- interface are completely different type and can have only public, static , final, constants and method declaration.
- Interface can only extend other interface.

Access Modifiers or Visibility of variables:

There are two types of modifiers in java: **access modifier** and **non-access modifier**. The access modifiers specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of access modifiers:

- private
- default (or friendly)
- protected
- public

There are many non-access modifiers such as

- static,
- abstract,
- synchronized,
- native,
- volatile,
- transient etc.

Here, we will learn access modifiers.

1) Private

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A
{
        private int data=40;
        private void msg()
        {
                System.out.println("Hello java");
        }
}
public class Simple
{
                 public static void main(String args[])
                {
   A obj=new A();
   obj.data);                          //Compile Time Error
   obj.msg();                          //Compile Time Error
   }
}
```

Role of Private Constructor:

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A
{
        private  A(){}//private constructor

        void msg(){System.out.println("Hello java");}
}

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

                A obj=new A();      //Compile Time Error
        }
}
```

Note: A class cannot be private or protected except nested class.

2) default

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within package. **Example of default access modifier**

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java

package pack;

class A
{
        void msg()
        {
                System.out.println("Hello");
        }
}
```

```
//save by B.java

package mypack;
import pack.*;

class B
{
        public static void main(String args[])
        {
                A obj = new A();   //Compile Time Error
                obj.msg();         //Compile Time Error
        }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package.

But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

//save by A.java

```
package pack;
public class A
{
        protected void msg()
        {
                System.out.println("Hello");
        }
}
```

```
//save by B.java

package mypack;
import pack.*;

class B extends A
{
        public static void main(String args[])
        {
                B obj = new B();
                obj.msg();
        }
}
```

Output:

Hello

4) public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java

package pack;
public class A
{
        public void msg()
        {
                System.out.println ("Hello");
        }
}
```

```
//save by B.java

package mypack;
import pack.*;

class B
{
  public static void main(String args[])
  {
                A obj = new A();
                obj.msg();
  }
}
```

Output:

Hello

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default or friendly	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Private protected Access:

This modifier makes the field visible in all subclasses of other package. But this field is not accessible by other class in the same package

Rules of Thumb to use access modifiers:

- Use public if the field is to be visible everywhere
- Use protected if the field is to be visible everywhere in the current package and also subclasses in the other package.
- Use “default” if the field is to be visible everywhere in the current package only.
- Use private protected if the field is to be visible only in subclass, regardless of package.
- Use private if the field is not to be visible anywhere except in its own class.

Array:

An array as a collection of variables of the same type.

Declaration:

dataType[ ] arrayRefVar; // preferred way.

or

dataType arrayRefVar[ ]; // works but not preferred way.

Example:

```
double[ ]   myList;   // preferred way.

  Or

double      myList[ ]; // works but not preferred way.
```

Declaring and Initializing an Arrays:

You can create an array by using the new operator with the following syntax:

Syntax:

```
dataType[ ]  arrayRefVar  =  new dataType[arraySize];
                              or
dataType[ ] arrayRefVar = {value0, value1, ..., valuek};
```

Example:

```
int [ ]  myList =  new int[5];
                  or
int [ ]  myList  =  {10,20,30,40,50};
```

```
myList [0] = 10
myList [1] = 20
myList [2] = 30
myList [3] = 40
myList [4] = 50
```

Note: The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Example:

```
public class array_ex
 {

        public static void main(String []args)
        {

                int arrex[] = {10,20,30};   //Declaring and initializing an array of three elements

                for (int i=0;i<arrex.length;i++)
                {

                        System.out.println(arrex[i]);

                }

        }

}
```

```
public class array_ex
{
        public static void main(String []args)
        {
                int arrex[] = new int[3]; //declaring array of three items
                        arrex[0] =10;
                        arrex[1] =20;
                        arrex[2] =30;
                //Printing array
                for (int i=0;i<arrex.length;i++)
                {
                                System.out.println(arrex[i]);
                }
        }
}
```

Example:

```
public class array_ex
{
        public static void main(String []args)
        {
            String strState[] = new String[3]; //declaring array of three items

                strState[0] ="New York";

                strState[1] ="Texas";

                strState[2] ="Florida";

          //Printing array

            for (int i=0;i<strState.length;i++)
            {

                System.out.println (strState[i]) ;

            }

        }

}
```

```
public class TestArray
{
        public static void main(String[] args)
        {
                double[] myList = {1.9, 2.9, 3.4, 3.5}; // Print all the array elements
                for (int i = 0; i < myList.length; i++)
                {
                        System.out.println(myList[i] + " ");
                }

                  // Summing all elements

                double total = 0;
                for (int i = 0; i < myList.length; i++)
                {
                        total += myList[i];
                }

                 // Finding the largest element
                System.out.println("Total is " + total);
                double max = myList[0];
                for (int i = 1; i < myList.length; i++)
                 {
                        if (myList[i] > max)
                        max = myList[i];
                }

                System.out.println("Max is " + max);
        }

}
```

Output:

1.9 2.9 3.4 3.5 Total is 11.7 Max is 3.5

```
// Sorting of Array Element and finding position of particular elements of an array.
import java.util.Arrays;
public class Sorting
{
        public static void main(String[] args)
       {
                int[] ar = new int[]{3,2,5,4,1};                    //create an int array

                System.out.print("Original Array : ");
                   for(int i=0; i < ar.length ; i++)
                                System.out.print("  "  + ar[i]);

                Arrays.sort(ar);       //To sort java int array use Arrays.sort()  method of java.util package.

                System.out.print("Sorted in array : ");

                for(int i=0; i < ar.length ; i++)
                                System.out.print("  "  + ar[i]);

// Searching a position of particular element
                    int searchVal = 4;
                    int retVal = Arrays.binarySearch(ar,   searchVal);
                   System.out.println("The index of element 4 is  : " + retVal);
        }
}
```

```
/*
        Output Would be
        Original Array :     3  2  5  4  1
        Sorted int array :     1  2  3  4  5
        The index of element 4 is : 3
 */
```

Write the program to find two arrays are equal or not:

```
import java.util.Arrays;
public class Main
{
                public static void main(String[] args) throws Exception
                {
                                int[] ary = {1,2,3,4,5,6};
                                 int[] ary1 = {1,2,3,4,5,6};
                                int[] ary2 = {1,2,3,4};

                                System.out.println("Is array 1 equal to array 2?? " +Arrays.equals(ary, ary1));

                                System.out.println("Is array 1 equal to array 3?? " +Arrays.equals(ary, ary2));
                 }
}
```

Output:

Is array 1 equal to array 2?? true
Is array 1 equal to array 3?? false

Multidimensional array :

it is used to store the data into table form or matrix form.

Declaration

```
int[ ][ ] num = new int[5][2];
        Or
int num[ ][ ] = new int[5][2];
```

Create a table with fifth row and 2 column:

	col [0]	col[1]
Row[0]		
Row[1]		
Row[2]		
Row[3]		
Row[4]		

Initialization

```
num[0][0] = 1;
num[0][1] = 2;
num[1][0] = 5;
 num[1][1] = 6;
num[2][0] = 11;
num[2][1] = 12;
 num[3][0] = 15;
 num[3][1] = 16;
num[4][0] = 18;
num[4][1] = 20;
```

	Col [0]	Col [1]
Row[0]	1	2
Row[1]	5	6
Row[2]	11	12
Row[3]	15	16
Row[4]	18	20

Or

```
int[ ][ ] num={ {1,2}, {5,6}, {11,12}, {15,16}, {18,20} };
```

For Accessing Two Dimensional Array:

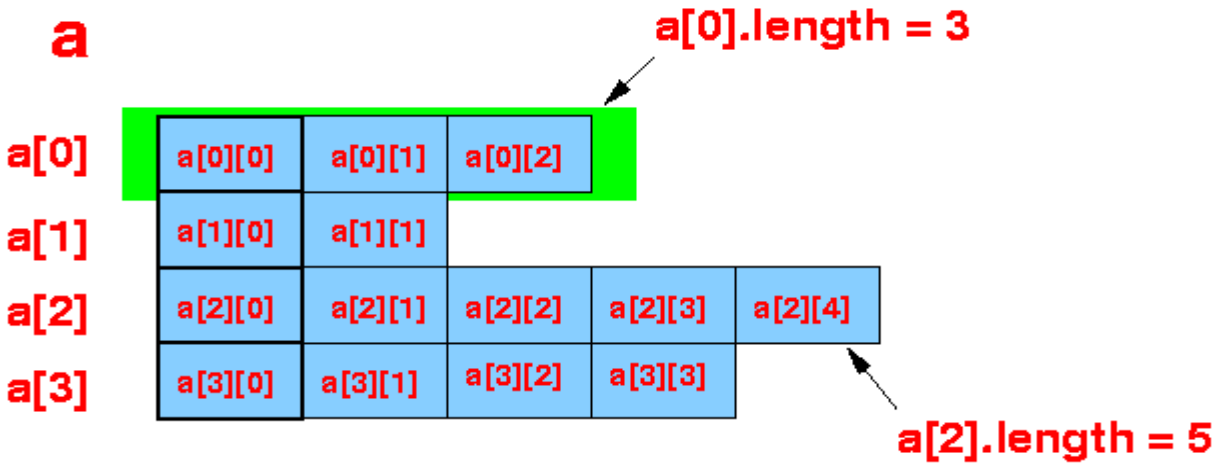
```
for (int i=0; i<(num.length); i++ )
{
        for (int j=0;j<num[i].length;j++)
        {
                System.out.println(num[i][j]);
        }
}
```

JAGGED ARRAY OR VARIABLE SIZE ARRAY:

ROW SIZE IS FIXED BUT COLUMN LENGTH IS DYNAMIC. IT IS CHANGED AS PER OUR DESIRED.

```
int[ ][ ] mul = new int[4][ ];

mul [0] = new int [3];
mul[1] = new int [2];
mul[2] = new int [5];
mul[3] = new int [4];
```

- it means that you have created a two dimensional array,
- with four rows,
- in first row there are 3 columns, second row 2 column, third row 5 column, fourth row 4 column.
- in java you can select column size for every row as your desire.



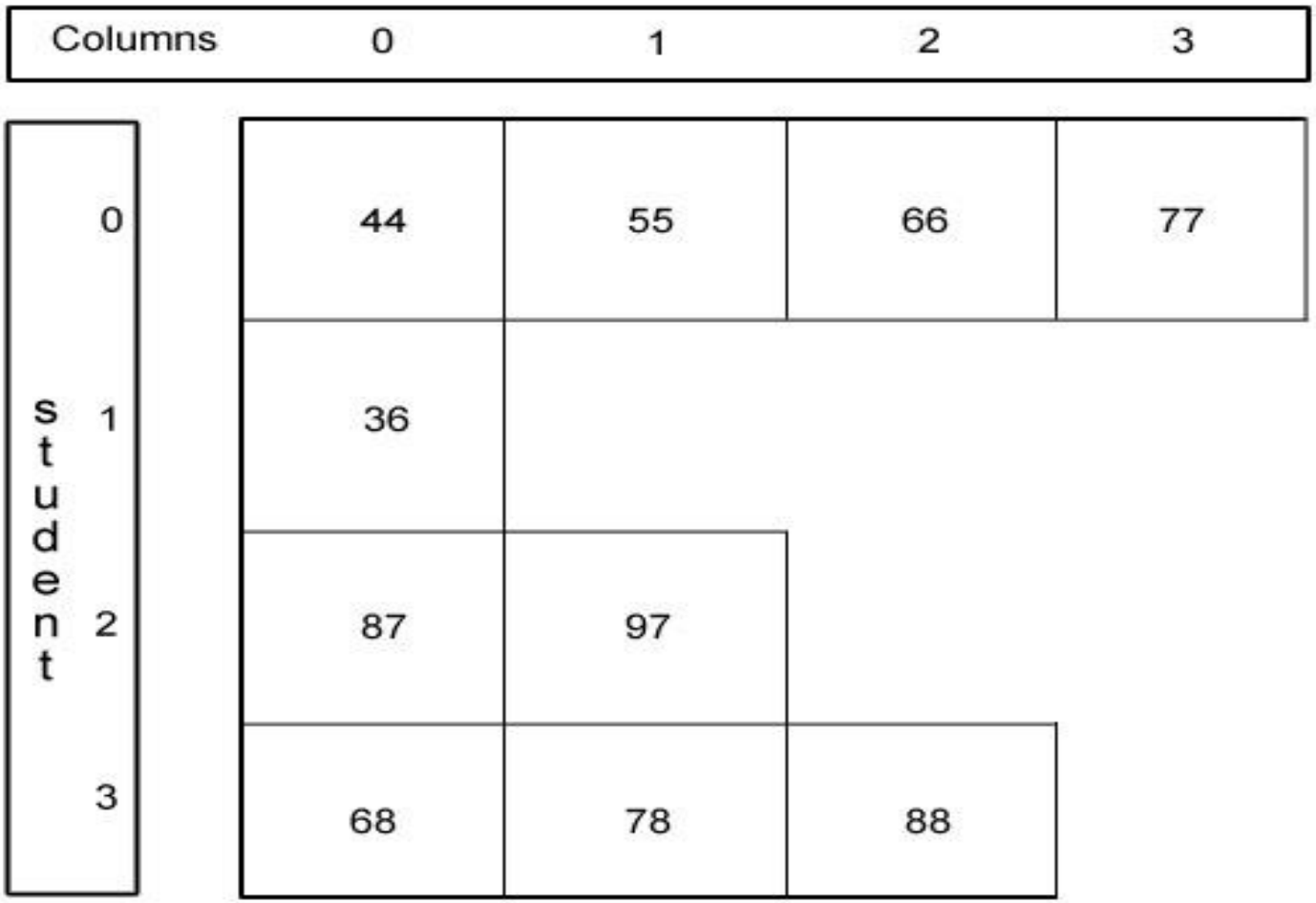
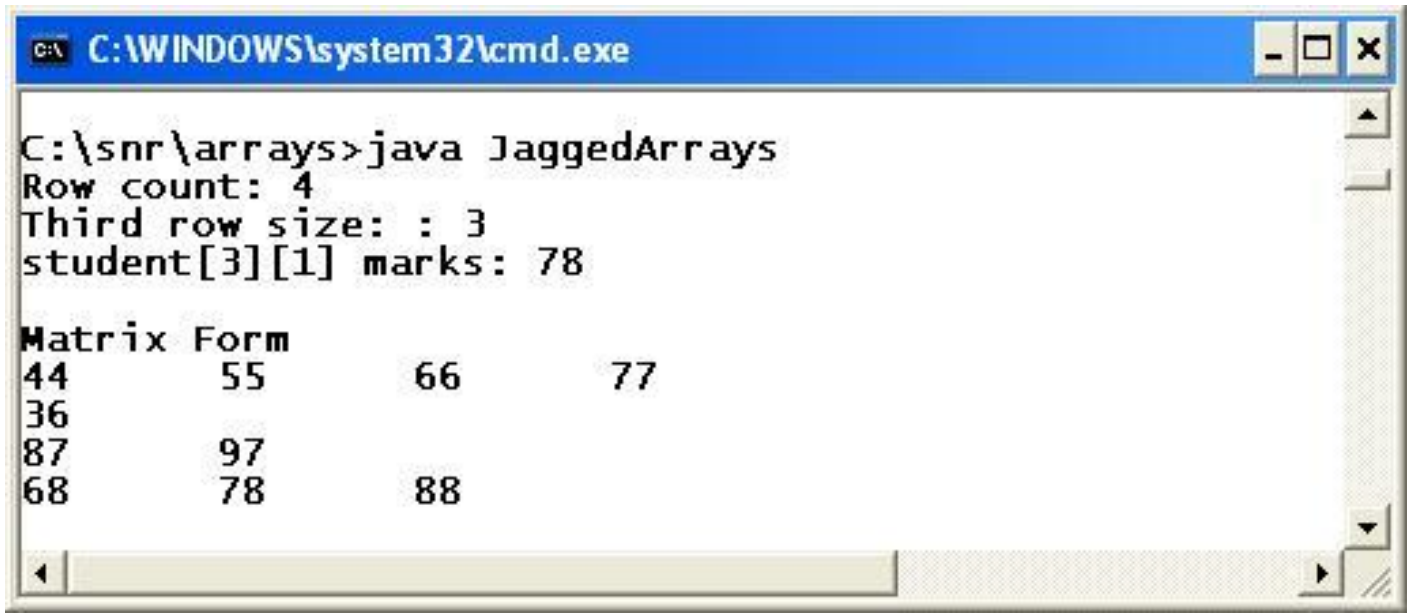


FIgure : Varying columns 2D array - matrix form

```
public class JaggedArrays
{
public static void main(String args[])
{
   int student[][] = new int[4][];
   student[0] = new int[4];
   student[1] = new int[1];
   student[2] = new int[2];
   student[3] = new int[3];

   System.out.println("Row count: " + student.length);
   System.out.println("Third row size: : " + student[3].length);
                          // 1st row
   student[0][0] = 44;
   student[0][1] = 55;
   student[0][2] = 66;
   student[0][3] = 77;
            // 2nd row
   student[1][0] = 36;
            // 3rd row
   student[2][0] = 87;
   student[2][1] = 97;
```

```
 // 4th row
   student[3][0] = 68;
   student[3][1] = 78;
   student[3][2] = 88;

System.out.println("student[3][1] marks: " + student[3][1]);

   System.out.println("\nMatrix Form");

   for(int i = 0; i < student.length; i++)
   {
     for(int j = 0; j < student[i].length; j++)
     {
       System.out.print(student[i][j] + "\t");
     }
     System.out.println();
   }
}
}
```


C:\WINDOWS\system32\cmd.exe
C:\snr\arrays>java JaggedArrays
Row count: 4
Third row size: : 3
student[3][1] marks: 78
Matrix Form
44 55 66 77
36
87 97
68 78 88

STRING

String is a sequence of character array.

In java, strings are class objects and implemented using two classes.

- String
- StringBuffer.

String class contain java.lang.String package.

Declaration of String:

String stringname = new String (“string”);

Example:

String firstname = new String (“Anil”) ;
String lastname = new String (“Sharma”) ;

In java + operator is used to concat two string.

String fullname = firstname + lastname ;
String city1 = “New” + “Delhi” ;

String Arrays:

String itemarray[ ] = new String[3] ;   // Here itemarray [ ] object can store 3 string.

Example:

```
   import java.util.Collections;
   import java.util.List;
   import java.util.Arrays;

 public class ReverseStringArrayExample
 {

                public static void main(String args[])
                {

                 //String array
                 String[] strDays = new String[]{"Sunday", "Monday", "Tuesday", "Wednesday"};

                //first create a list from String array
                 List<String> list = Arrays.asList(strDays);

                 //next, reverse the list using Collections.reverse method
                 Collections.reverse(list);

                 //next, convert the list back to String array
                 strDays = (String[]) list.toArray();
```

```
 System.out.println("String array reversed");

            //print the reversed String array
            for(int i=0; i < strDays.length; i++){
                  System.out.println(strDays[i]);
            }

      }

}


/*
Output of above given Java Reverse String Array example would be
String array reversed
Wednesday
Tuesday
Monday
Sunday
*/
```

String Methods:

Following table shows some string methods:

S2 = s1.toLowerCase	Convert the string s1 to all lowercase
S2 = s1.toUpperCase	Convert the string s1 to all uppsercase
S2 = s1.replace (‘x’ , ‘y’)	Replace all appearance of x with y.
S2 = s1.trim ()	Remove white space at the beginning and end of string s1.
S1.equals (s2)	Returns ‘true ’ if string s1 equals string s2.
S1.equalIgnoreCase (s2)	Returns ‘true ’ if string s1 equals string s2, ignoring the case of character.
S1.length ()	Gives the length of S1
S1.charAt (n)	Gives the nth character of S1
S1.compareTo (s2)	Returns negative if s1 < s2, positive if s1 > s2 and zero if s1 equal to s2.
S1.concat (s2)	Concatenates s1 and s2
S1.substring (n)	Gives substring starting from the nth character
S1.substring (n , m)	Give substring starting from nth character up to mth character.
String.Valueof (p)	Create a string object of the parameter p (simple type or object)
p.toString ()	Create a string representation of object p.
S1.indexOf (‘x’)	Gives the position of the first occurrence of ‘x’ in the string s1
S1.indexOf (‘x’, n)	Gives the position of ‘x’ that occurs after nth position in string s1.

Example of valueOf ()

```
import java.io.*;
public class Test
{
        public static void main(String args[])
        {
                double d = 102939939.939;
                boolean b = true;
                long l = 1232874;
                char[] arr = {'a', 'b', 'c', 'd', 'e', 'f','g' };

                System.out.println("Return Value : " + String.valueOf(d) );
                System.out.println("Return Value : " + String.valueOf(b) );
                System.out.println("Return Value : " + String.valueOf(l) );
                System.out.println("Return Value : " + String.valueOf(arr) );
        }
 }
```

Output:

```
Return Value : 1.02939939939E8
Return Value : true
Return Value : 1232874
Return Value : abcdefg
```

Example of toString ()

```
import java.io.*;
public class Test
{
        public static void main(String args[])
        {
                String Str = new String("Welcome to Tutorialspoint.com");
                System.out.print("Return Value :");
                System.out.println(Str.toString());
        }
}
```

Output:

```
Welcome to Tutorialspoint.com
```

Example of charAt()

```
public class Test
{
        public static void main(String args[])
        {
                String s = "Strings are immutable";
                char result = s.charAt(8);
                System.out.println(result);
        }
}
```

StringBuffer Class:

String class creates string of fixed length, StringBuffer class creates string of variable length or dynamic length that can be modified in terms of content and length.

Example: We can insert characters and substring in the middle of the string or append another string at the end of the string.

Commonly used method of StringBuffer class is as per the following:

Method	Task
S1.setCharAt(n, 'x')	Modifies the nth character to 'x'
S1.append (s2)	Append the string s2 to s1 at the end
S1.insert (n, s2)	Insert the string s2 at the position n of string s1
S1.setLength (n)	Sets the length of the string s1 to n. if n < s1.length s1 is truncated. If n>s1.length zeros are added to s1.
delete(int start, int end)	Removes the characters in a substring of this StringBuffe
replace(int start, int end, String str)	Replaces the characters in a substring of this StringBuffer with characters in the specified String.
reverse()	The character sequence contained in this string buffer is replaced by the reverse of the sequence.
setLength(int newLength)	Sets the length of this String buffer.

```
import  java.lang.*;
class stringbuffer1
{
                public static void main (String args[])
               {
                              StringBuffer str = new StringBuffer ("Object Lanaguage");

                              System.out.println ("Length of the string is :" + str.length());

                              // Inserting string in the middle

                              str.insert (7, "Oriented ");
                              System.out.println ("Modified string is " + str);


                              // Modifying character
                              str.setCharAt (6, '-');
                              System.out.println ("Modified string is " + str);

                              // Append a string at the end of string
                              str.append ("improves security");
                              System.out.println ("Modified string is " + str);

                              // Reverse String is
                              str.reverse ();
                              System.out.println ("Reverse string is " + str);
                }

}
```