**History of JAVA**

JAVA is a distributed technology developed by James Gosling, Patric Naugton, etc., at Sun Micro System has released lot of rules for JAVA and those rules are implemented by JavaSoft Inc, USA (which is the software division of Sun Micro System) in the year 1990. The original name of JAVA is OAK (which is a tree name). In the year 1995, OAK was revised and developed software called JAVA (which is a coffee seed name).

JAVA released to the market in three categories J2SE (JAVA 2 Standard Edition), J2EE (JAVA 2 Enterprise Edition) and J2ME (JAVA 2 Micro/Mobile Edition).

1. J2SE is basically used for developing client side applications/programs.
2. J2EE is used for developing server side applications/programs.
3. J2ME is used for developing server side applications/programs.

If you exchange the data between client and server programs (J2SE and J2EE), by default JAVA is having on internal support with a protocol called http. J2ME is used for developing mobile applications and lower/system level applications. To develop J2ME applications we must use a protocol called WAP (Wireless Applications Protocol).

**FEATURES of java**

1. Simple
2. Platform independent
3. Architectural neutral
4. Portable
5. Multi threading
6. Distributed
7. Networked
8. Robust
9. Dynamic
10. Secured
11. High performance
12. Interpreted
13. Object Oriented Programming Language
14. **Simple**: JAVA is simple because of the following factors:
    * JAVA is free from pointers hence we can achieve less development time and less Execution time [whenever we write a JAVA program we write without pointers and Internally it is converted into the equivalent pointer program].
    * Rich set of API (application protocol interface) is available to develop any complex Application.
    * The software JAVA contains a program called garbage collector which is always used to Collect unreferenced (unused) memory location for improving performance of a JAVA Program. [Garbage collector is the system JAVA program which runs in the background Along with regular JAVA program to collect unreferenced memory locations by running At periodical interval of times for improving performance of JAVA applications.
    * JAVA contains user friendly syntax’s for developing JAVA applications.
15. **Platform Independent:**

A program or technology is said to be platform independent if and only if which can run on all available operating systems.

The languages like C, Cpp are treated as platform dependent languages since these

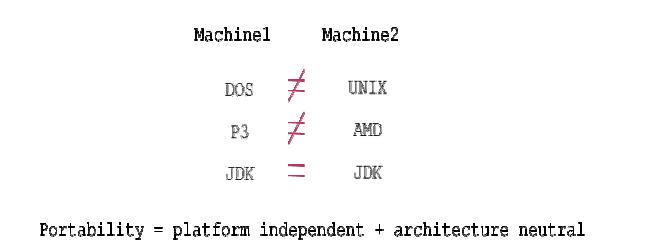
46

languages are taking various amount of memory spaces on various operating systems [the operating system dos understands everything in the form of its native format called Mozart (MZ) whereas the operating system Unix understands everything in its negative format called embedded linking format (elf). When we write a C or Cpp program on dos operating and if we try to transfer that program to Unix operating system, we are unable to execute since the format of these operating systems are different and more over the C, Cpp software does not contain any special programs which converts one format of one operating system to another format of other operating system].

The language like JAVA will have a common data types and the common memory spaces on all operating systems and the JAVA software contains the special programs which converts the format of one operating system to another format of other operating system. Hence JAVA language is treated as platform independent language.

3. Architectural Neutral:

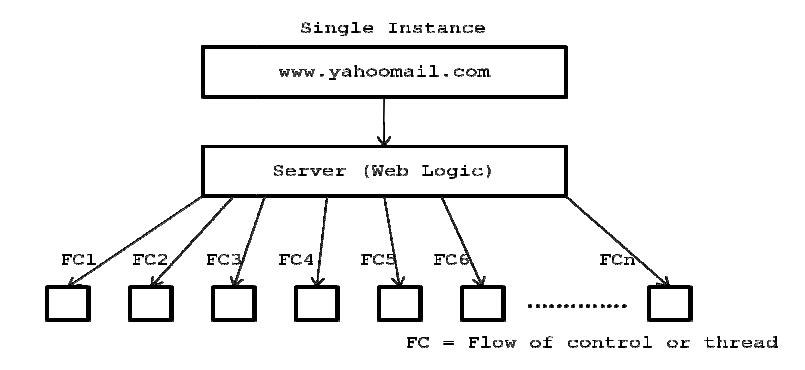
A language or technology is said to be architectural neutral which can run on any available processors in the real world. The languages like C, Cpp are treated as architectural dependent. The language like JAVA can run on any of the processor irrespective of their Vendor.



**4. Portable:**

A portable language is one which can run on all operating systems and on all processors irrespective their architectures and providers. The languages like C, Cpp are treated as non portable languages whereas the language JAVA is called portable language.

**5. Multi Threading:**



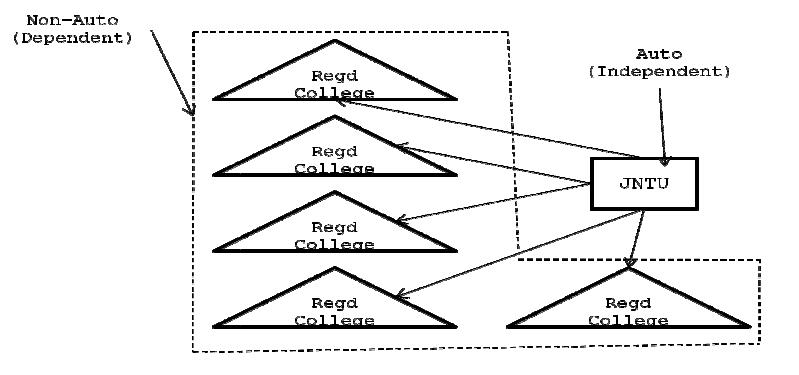
* A flow of control is known as thread
* A multi threaded program is one in which there exists multiple flow of controls i.e. threads

47

* + A program is said to be multi threaded program if and only of there exists n number of sub-programs. For each and every sub sub-program there exists a separate flow of control. All such flow of controls are executing concurrently. Such flow of controls is known as threads .Such type of applications type of applications is known as multithreading applications
  + The languages like C, Cpp are treated as single threaded modeling languages (STML). SMTL are those in which there exists single flow of control
  + The languages like JAVA and DOT NET are treated as multi threaded modeling languages(MTML). MTML are those in which there exist multiple flows of controls

1. Distributed:

A service is a said to be a distributed service which runs in multiple servers and that service can be accessed by n number of clients across the globe. In order to develop distributed applications we must require architecture called trusted network architecture. To develop these applications we require a technology called J2EE. Distributed applications are preffered by larger scale organizations.



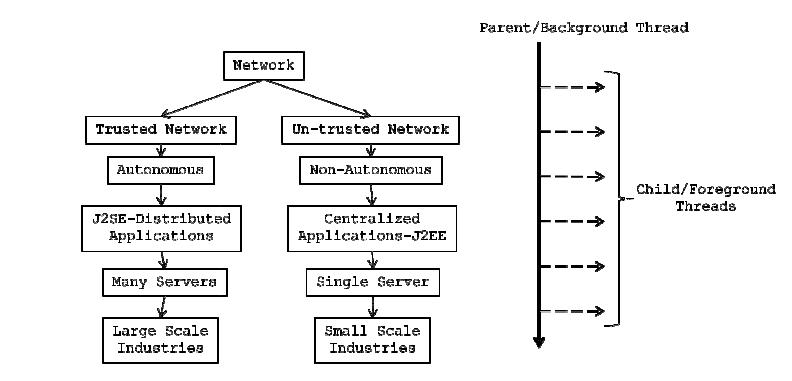
**7. Networked:**

In real world we have two types of networks. They are un-trusted network and trusted networks.

**Un-trusted networks:**

A network is said to be un-trusted network in which there exists n number of inter connected non-autonomous architecture Un-trusted network is also known as LAN.Using this network architecture, we develop centralized applications. A centralized application is one which runs on single server and it can be access in limited graces. In order to develop centralized applications we may use a technology called J2SE and these kinds of applications are preferred by small scale organization.

48



**Trusted network:**

A network is said to be trusted network in which there exists n number of inter connected autonomous architecture. Trusted network Trusted network is also know as WAN Using this network, we can develop distributed applications .A distributed application is one which runs on multipul servers and it can be access in unlimited graces. In order to develop distributed applications we may use a technology called J2EE and these kinds of applications are preferred by large scale organizations.

## Java Installation

Once java is downloaded, it can be installed like any other software (**.exe**) in your Windows system

Setting up the Environment Variables

After installing Java there are some environment variables that need to be set.

* **CLASSPATH :** This environment variable points the location of JDK home directory. It also contains the address of the folder from where the jars get loaded by the ClassLoader (For more details of ClassLoader visit here)
* **JAVA\_HOME** : This environment variable will point the location of Java home directory.
* **Windows Vista / 7**
* To set up environment variables in Windows Vista / 7  right click  on the “Computer” and select Properties. In the Property window select the “ADVANCED SYSTEM SETTINGS” and then select the “ADVANCED” tab and click “ENVIRONMENT VARIABLES”. A window will appear were you can enter a new environment variable under User/System Variables by clicking on the New button.

## How to Check if Java is Installed

To check if your java is installed properly open Command Prompt . To open command prompt write “CMD” in run command and hit enter. In the command prompt window write “***java -version***“.  If your java is installed properly and all environment variables are configured correctly it will show the version of Java installed . Information reflected on the command prompt will be like

JDK (Java Development Kit)

It contains everything that will be required to***develop and run*** any Java Application.

JRE (Java Run time Environment)

It contains everything required to ***run*** any Java Application which is already compiled. It doesn’t contain library which is required to develope java application.

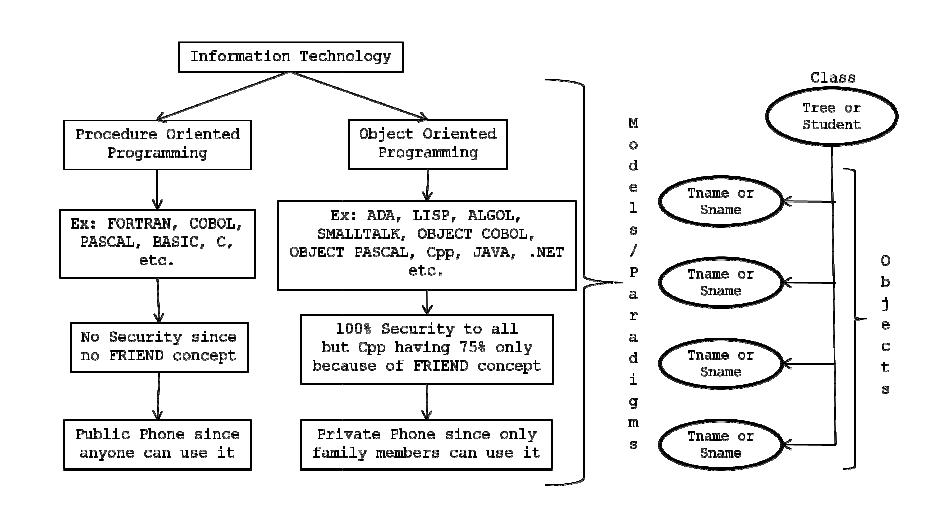
JVM (Java Virtual Machine)

It is a virtual machine which work over your operating system to provide proper environment for your compiled Java code. JVM only works with bytecode. Hence you need to compile your java application(.java) so that it can be converted to bytecode format (.class file). Which then will be used by JVM to run application. JVM only provide environment it needs other library to run application properly.

**8.Java is OBJECT ORIENTED PROGRAMMING**

In an IT we have two types of programming models (pa paradigms) are available. They are procedure oriented programming language and object oriented programming language

If we represent the data using procedural oriented programming languages then there is no security for the data which we represent . For example when we represent the data of a student in C language using structures concept, the student data can be accessed by all the functions which we write as a part of C program. If one of the functions manipulates or damages the data then we are loosing correction-less (integrity) of the data. Examples of procedure oriented programming languages are FORTRON, COBOL, PASCAL, BASIC, C, etc.



When we represent the data in object oriented programming language we get the security.Examples of object oriented programming languages are LISP, ADA, ALGOL,

49

SMALLTALK, OBJECT,COBOL, OBJECT PASCAL, Cpp, JAVA , DOT NET, etc. In order to say any language is an object oriented programming language.

**OOPs Principles**

1. Class
2. Object
3. Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism
7. Interfaces
8. Packages

**Setup-Environment variables**

* Right Click on my computer icon of desktop of your computer
* Select properties
* Select Advanced tab
* click on environment variables button
* Click on New/Edit to create a variable

Here the variable is a pair of values name & value

* Path:c:\programfiles\java\JDK1.6.0.1\bin
* ClassPath(JavaFiles): %classpath%path for java files not spaces

**Eclipse**

It is one of the editors for writing the Java code.

**Install eclipse.exe:**

1. Download Eclipse Standard 4.3.1 from web.

2. Install Eclipse by double click on eclipse.exe.

3. Give project location as Workspac





**Creating a Java project:**

1. Goto to Workbench

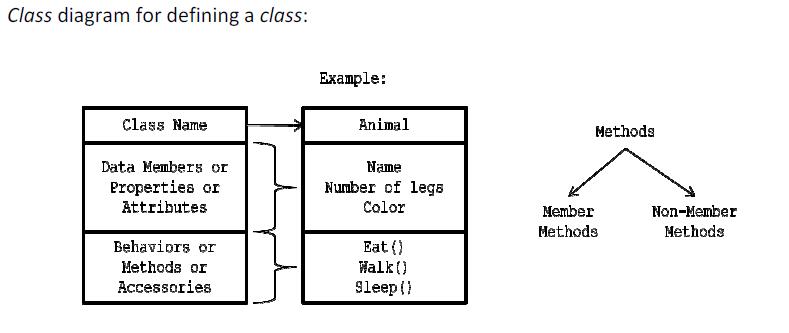
2. Click on File > New > Java Project



3.Choose your JRE version, latest being JavaSE – 1.7

4. Click on Finish

**Class :** A class can be defined as a template/ blue print that describe the behaviors/states that objectof its type support. (or) “A class is a way of binding the data and associated methods in a single unit



**Syntax for defining a CLASS:**

Class <clsname>

{

Variable declaration; Methods definition; };

50

Here, class is a **keyword** which is used for developing or creating **user defined data types** Clsname represents a JAVA valid variable name and it is treated as name of the class. Class names are used for creating **objects.**

Class contains two parts namely **variable declaration** and **method definitions**. Variable Declaration represents what type of **data members** which we use as a part of the class. Method definition represents the type of methods which we used as the path of the class to perform an operation.

By making use of the variables, which are declared inside the class? Every operation in JAVA must be defined with in the class only i.e. outside definition is not possible.

**First Java Program to print Hello World** public class MyFirstJavaProgram

{

/\* This is my first java program.

\* This will print 'Hello World' as the output \*/

public static void main(String []args)

{

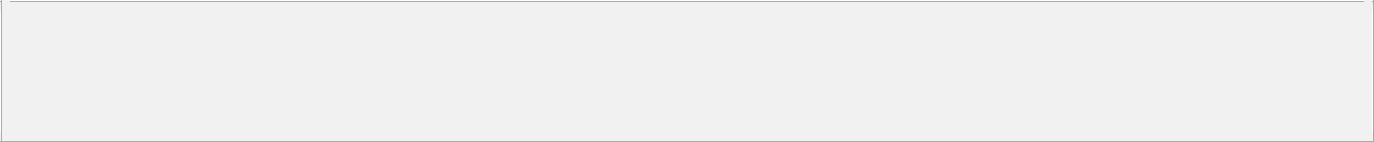
System.out.println("Hello World"); // prints Hello World

}

}

**Lets look at how to save the file, compile and run the program. Please follow the steps given below:**

1. Open notepad and add the code as above.
2. Save the file as : MyFirstJavaProgram.java.
3. Open a command prompt window and go o the directory where you saved the class. Assume its C:\.
4. Type ' javac MyFirstJavaProgram.java ' and press enter to compile your code. If there are no errors in your code the command prompt will take you to the next line.( Assumption : The path variable is set).
5. Now type ' java MyFirstJavaProgram ' to run your program.
6. You will be able to see ' Hello World ' printed on the window.

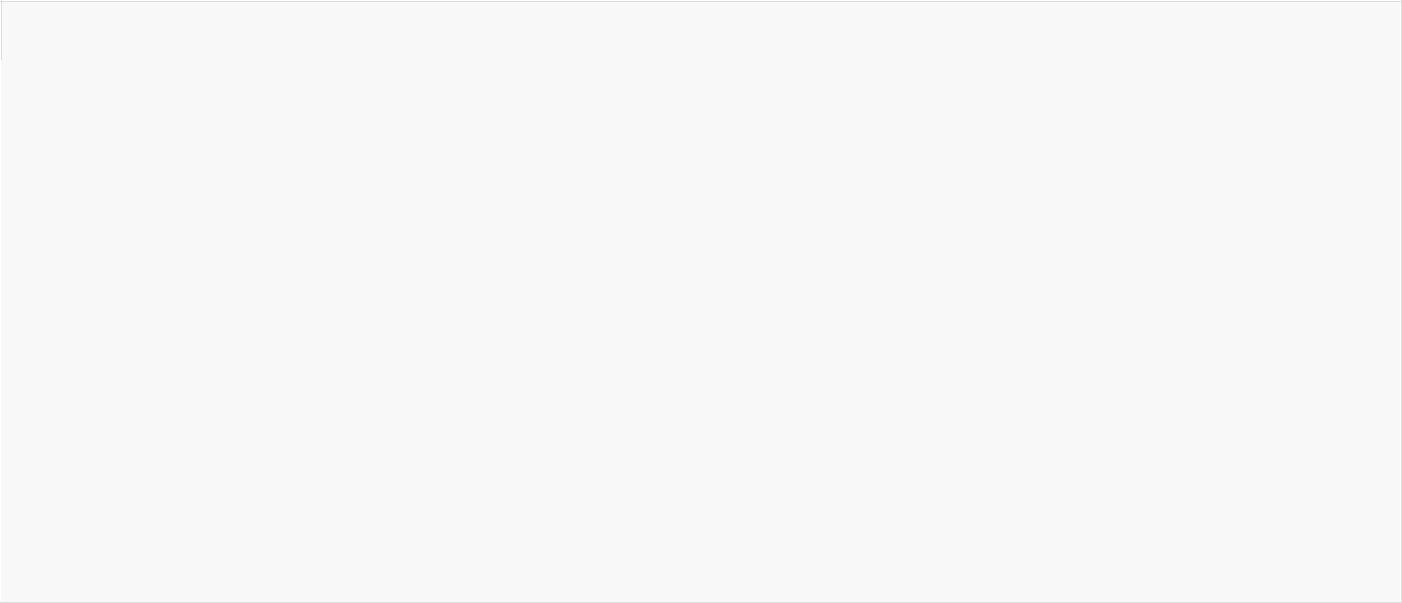


C : > javac MyFirstJavaProgram.java

C : > java MyFirstJavaProgram

Hello World

**Program for taking the inputs from key board(Addition of 2 numbers)**



import java.util.Scanner; class AddNumbers

{

public static void main(String args[])

{

int x, y, z;

System.out.println("Enter two integers to calculate their sum ");

Scanner in = new Scanner(System.in);

x = in.nextInt(); y = in.nextInt(); z = x + y;

System.out.println("Sum of entered integers = "+z);

}

}

51

**About Java programs, it is very important to keep in mind the following points**.

**Case Sensitivity** - Java is case sensitive which means identifier Hello and hello would have differentmeaning in Java.

**Class Names** - For all class names the first letter should be in Upper Case.

If several words are used to form a name of the class each inner words first letter should be in Upper Case. Example class MyFirstJavaClass

**Method Names** - All method names should start with a Lower Case letter.

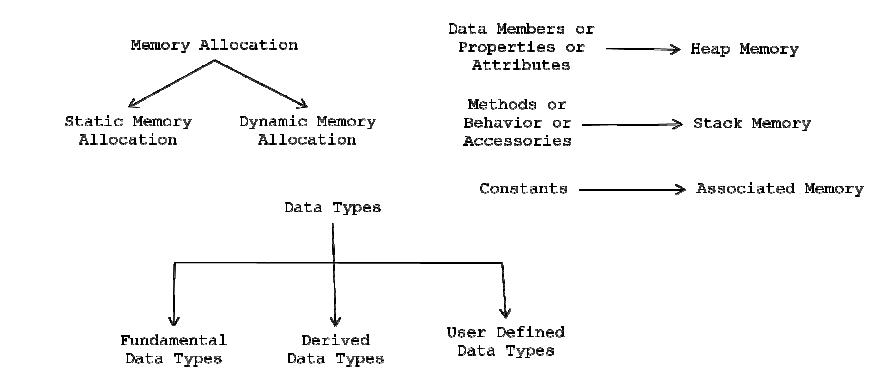
If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. Example public void myMethodName()

**Program File Name** - Name of the program file should exactly match the class name.Example : Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'

**public static void main(String args[])** - java program processing starts from the main() method whichis a mandatory part of every java program..

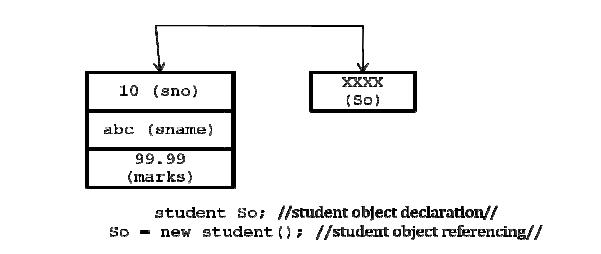
**Object** - Objects have states and behaviors. Example: A dog has states-color, name, breed as well asbehaviors -wagging, barking, eating. An object is an instance of a class. (or)

**Instance** (instance is a mechanism of allocating sufficient amount of memory space for datamembers of a class) of a class is known as an object.



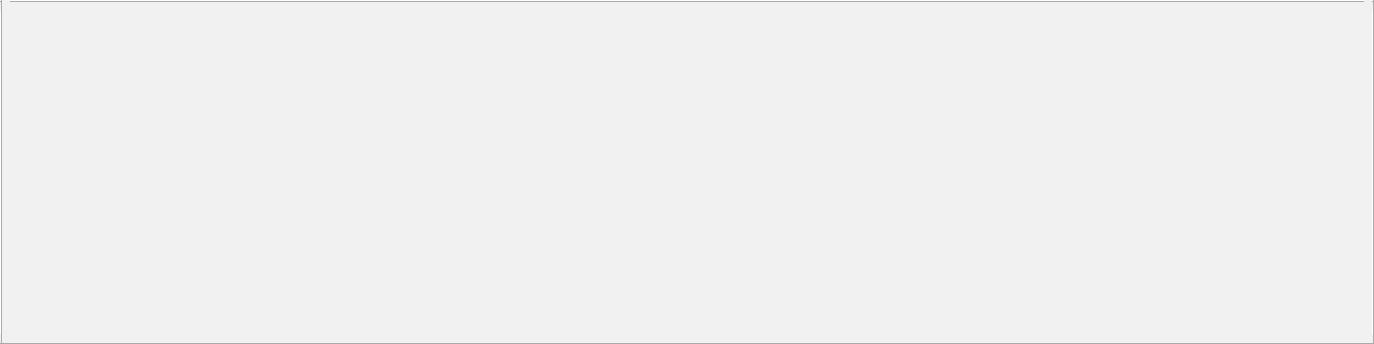
**Syntax-1 for defining an OBJECT:**

<Clsname> objname = new <clsname ()> Student so = new student();



**Example of creating an object is given below:**

52



class Puppy{

public Puppy(String name)

{

System.out.println("Passed Name is :" + name );

}

public static void main(String []args){

// Following statement would create an object myPuppy Puppy myPuppy = new Puppy( "tommy" );

}

}

If we compile and run the above program then it would produce following result:

Passed Name is :tommy



**Main Class:** It is important section for all programming .it is defined on a class which as main methodis called main class.

**Syntax:**

Public static void main (String args[])

{

}

**Data Types**

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

**There are two data types available in Java:**

1. Primitive Data Types
2. Reference/Object Data Types

**Primitive Data Types:**

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

**byte:**

* Byte data type is a 8-bit signed two.s complement integer.
* Default value is 0
* Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
* Example : byte a = 100 , byte b = -50

**short:**

* Short data type is a 16-bit signed two's complement integer.
* Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
* Default value is 0.
* Example : short s= 10000 , short r = -20000

53

**int:**

* Int data type is a 32-bit signed two's complement integer.
* Int is generally used as the default data type for integral values unless there is a concern about memory.
* The default value is 0.
* Example : int a = 100000, int b = -200000

**long:**

* Long data type is a 64-bit signed two's complement integer.
* This type is used when a wider range than int is needed.
* Default value is 0L.
* Example : int a = 100000L, int b = -200000L

**float:**

* Float data type is a single-precision 32-bit IEEE 754 floating point.
* Float is mainly used to save memory in large arrays of floating point numbers.
* Default value is 0.0f.
* Float data type is never used for precise values such as currency.
* Example : float f1 = 234.5f

**double:**

* double data type is a double-precision 64-bit IEEE 754 floating point.
* This data type is generally used as the default data type for decimal values. generally the default choice.
* Double data type should never be used for precise values such as currency.
* Default value is 0.0d.
* Example : double d1 = 123.4

**boolean:**

* boolean data type represents one bit of information.
* There are only two possible values : true and false.
* This data type is used for simple flags that track true/false conditions.
* Default value is false.
* Example : boolean one = true

**char:**

* char data type is a single 16-bit Unicode character.
* Minimum value is '\u0000' (or 0).
* Maximum value is '\uffff' (or 65,535 inclusive).
* Char data type is used to store any character.
* Example . char letterA ='A'

**Reference Data Types:**

* Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
* Class objects, and various type of array variables come under reference data type.
* Default value of any reference variable is null.

54

* A reference variable can be used to refer to any object of the declared type or any compatible type.
* Example : Animal animal = new Animal("giraffe");
* Java language supports few special escape sequences for String and char literals as well. They

are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Notation** |  | **Character represented** |  |
|  |  |  |
|  | \n |  | Newline (0x0a) |  |
|  | \f |  | Formfeed (0x0c) |  |
|  |  |  |
|  | \b |  | Backspace (0x08) |  |
|  |  |  |
|  | \s |  | Space (0x20) |  |
|  |  |  |
|  |  |  |  |  |
|  | \t |  | tab |  |
|  | \" |  | Double quote |  |
|  |  |  |
|  | \' |  | Single quote |  |
|  |  |  |
|  | \\ |  | backslash |  |
|  |  |  |
|  | \ddd |  | Octal character (ddd) |  |
|  |  |  |
|  | \uxxxx |  | Hexadecimal UNICODE character (xxxx) |  |
|  |  |  |
|  |  |  |  |  |

Eg:

Public class example1

{

Public static void main (string args[])

{

Int a=10; Int b=20;

System.out.println(a+b);

}

}

**Variable Types**

In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

type identifier [ = value][, identifier [= value] ...] ;



The type is one of Java's datatypes. The identifier is the name of the variable. To declare more than one variable of the specified type, use a comma-separated list.

Here are several examples of variable declarations of various types. Note that some include an initialization.

|  |  |  |  |
| --- | --- | --- | --- |
| int a, b, c; | // declares | three | ints, a, b, and c. |
| int d = 3, e, f = 5; | // declares | three | more ints, initializing |
|  | // d and f. |  |  |
| byte z = 22; | // initializes z. | |  |
| double pi = 3.14159; | // declares | an approximation of pi. | |
| char x = 'x'; | // the variable x | | has the value 'x'. |
|  |  |  |  |

55

**There are three kinds of variables in Java:**

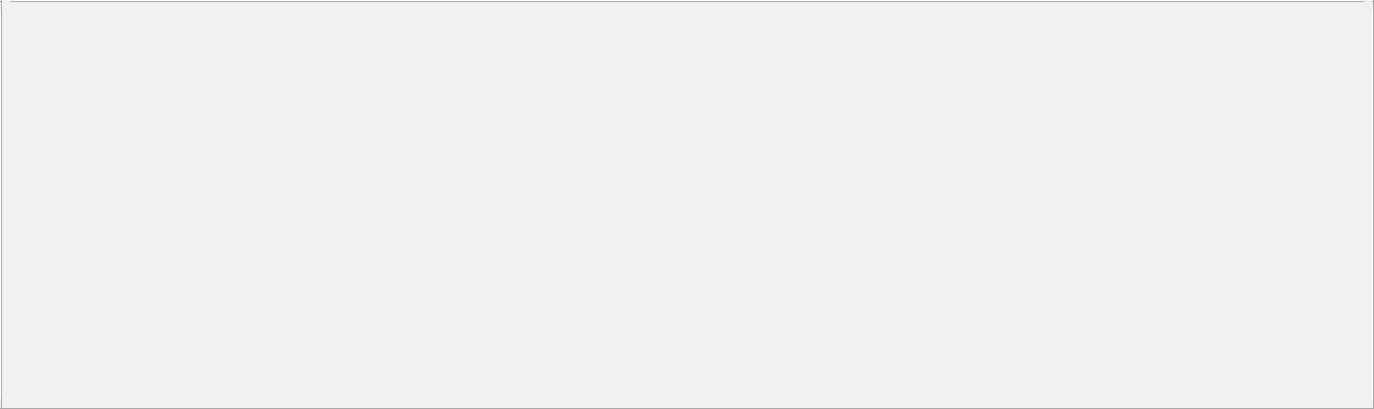
1. Local variables
2. Instance variables
3. Class/static variables

**Local variables :**

* Local variables are declared in methods, constructors, or blocks.
* Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
* Access modifiers cannot be used for local variables.
* Local variables are visible only within the declared method, constructor or block.
* Local variables are implemented at stack level internally.
* There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

**Example:**

Here age is a local variable. This is defined inside pupAge() method and its scope is limited to this method only.



public class Test{ public void pupAge(){

int age = 0; age = age + 7;

System.out.println("Puppy age is : " + age)

}

public static void main(String args[]){ Test test = new Test(); Test.pupAge();

}

}

This would produce following result:

Puppy age is: 7



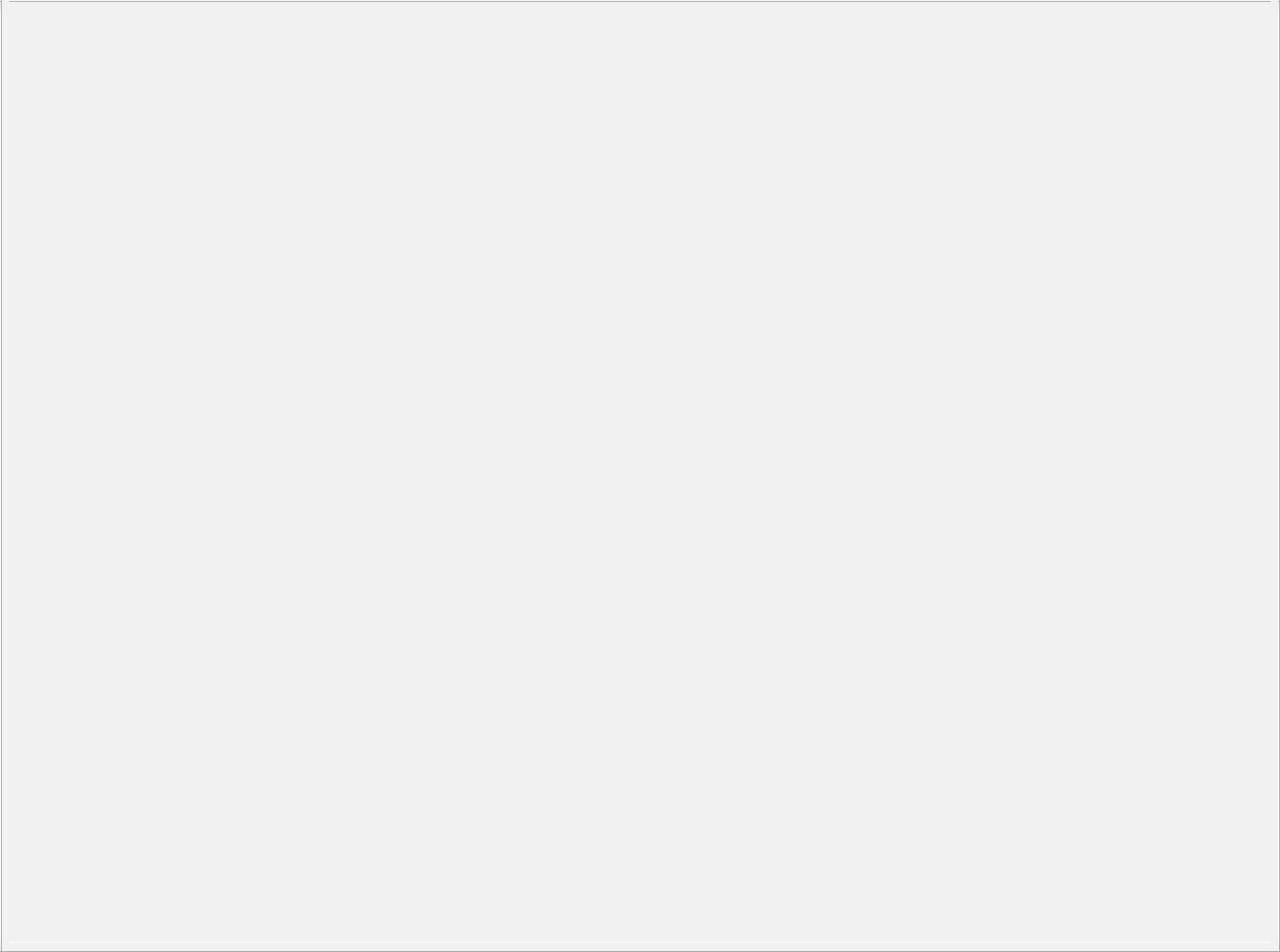
**Instance variables :**

* Instance variables are declared in a class, but outside a method, constructor or any block.
* When a space is allocated for an object in the heap a slot for each instance variable value is created.
* Instance variables are created when an object is created with the use of the key word 'new' and destroyed when the object is destroyed.
* Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object.s state that must be present through out the class.
* Instance variables can be declared in class level before or after use.

56

* Access modifiers can be given for instance variables.
* The instance variables are visible for all methods, constructors and block in the class. Normally it is recommended to make these variables private (access level).However visibility for subclasses can be given for these variables with the use of access modifiers.
* Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.
* Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class ( when instance variables are given accessibility) the should be called using the fully qualified name . ObjectReference.VariableName.

**Example:**



import java.io.\*;

class Employee{

* this instance variable is visible for any child class. public String name;
* salary variable is visible in Employee class only. private double salary;
* The name variable is assigned in the constructor. public Employee (String empName){

name = empName;

}

* The salary variable is assigned a value.

public void setSalary(double empSal){ salary = empSal;

}

// This method prints the employee details. public void printEmp(){

System.out.println("name : " + name );

System.out.println("salary :" + salary);

}

public static void main(String args[]){ Employee empOne = new Employee("Ransika"); empOne.setSalary(1000);

empOne.printEmp();

}

}

This would produce following result:



name : Ransika salary :1000.0

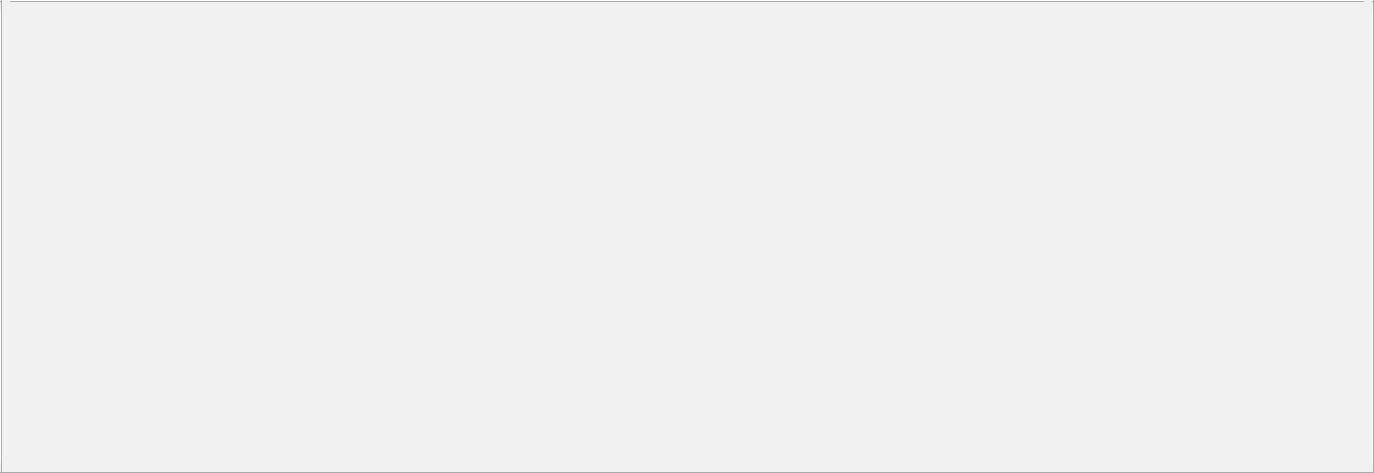
**Class/static variables :**

* Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
* There would only be one copy of each class variable per class, regardless of how many objects are created from it.

57

* Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final and static. Constant variables never change from their initial value.
* Static variables are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.
* Static variables are created when the program starts and destroyed when the program stops.
* Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
* Default values are same as instance variables. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor. Additionally values can be assigned in special static initializer blocks.
* Static variables can be accessed by calling with the class name . ClassName.VariableName.
* When declaring class variables as public static final, then variables names (constants) are all in upper case. If the static variables are not public and final the naming syntax is the same as instance and local variables.

**Example:**



import java.io.\*;

class Employee1{

* salary variable is a private static variable private static double salary;
* DEPARTMENT is a constant

public static final String DEPARTMENT = "Development";

public static void main(String args[]){ salary = 1000;

System.out.println(DEPARTMENT+"average salary:"+salary);

}

}

This would produce following result:

Development average salary:1000



**Java Access Modifiers**

The Java language has a wide variety of modifiers, including the following:

* Java Access Modifiers
* Non Access Modifiers

Java provides a number of **access modifiers** to set access levels for classes, variables, methods and constructors. The four access levels are:

1. Visible to the package. the **default**. No modifiers are needed.
2. Visible to the class only (**private**).
3. Visible to the world (**public**).
4. Visible to the package and all subclasses (**protected**).

Java provides a number of **non-access modifiers** to achieve many other functionality.

* The **static** modifier for creating class methods and variables
* The **final** modifier for finalizing the implementations of classes, methods, and variables.
* The **abstract** modifier for creating abstract classes and methods.
* The synchronized and volatile modifiers, which are used for threads.

# **Access Modifiers in java**

1. [private access modifier](http://www.javatpoint.com/access-modifiers#accessprivate)
2. [Role of private constructor](http://www.javatpoint.com/access-modifiers#accessprivatecons)
3. [default access modifier](http://www.javatpoint.com/access-modifiers#accessdefault)
4. [protected access modifier](http://www.javatpoint.com/access-modifiers#accessprotected)
5. [public access modifier](http://www.javatpoint.com/access-modifiers#accesspublic)
6. [Applying access modifier with method overriding](http://www.javatpoint.com/access-modifiers#accessoverriding)

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

### **1) private access modifier**

|  |
| --- |
| 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. |

1. **class** A{
2. **private** **int** data=40;
3. **private** **void** msg(){System.out.println("Hello java");}
4. }
6. **public** **class** Simple{
7. **public** **static** **void** main(String args[]){
8. A obj=**new** A();
9. System.out.println(obj.data);//Compile Time Error
10. obj.msg();//Compile Time Error
11. }
12. }

### 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: |

1. **class** A{
2. **private** A(){}//private constructor
3. **void** msg(){System.out.println("Hello java");}
4. }
5. **public** **class** Simple{
6. **public** **static** **void** main(String args[]){
7. A obj=**new** A();//Compile Time Error
8. }
9. }

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

### **2) default access modifier**

|  |
| --- |
| 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. |

1. //save by A.java
2. **package** pack;
3. **class** A{
4. **void** msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. **package** mypack;
8. **import** pack.\*;
9. **class** B{
10. **public** **static** **void** main(String args[]){
11. A obj = **new** A();//Compile Time Error
12. obj.msg();//Compile Time Error
13. }
14. }

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 access modifier**

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.

1. //save by A.java
2. **package** pack;
3. **public** **class** A{
4. **protected** **void** msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. **package** mypack;
8. **import** pack.\*;
10. **class** B **extends** A{
11. **public** **static** **void** main(String args[]){
12. B obj = **new** B();
13. obj.msg();
14. }
15. }

Output:Hello

### **4) public access modifier**

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

### **Example of public access modifier**

1. //save by A.java
3. **package** pack;
4. **public** **class** A{
5. **public** **void** msg(){System.out.println("Hello");}
6. }
7. //save by B.java
9. **package** mypack;
10. **import** pack.\*;
12. **class** B{
13. **public** **static** **void** main(String args[]){
14. A obj = **new** A();
15. obj.msg();
16. }
17. }

Output:Hello

### **Understanding all java access modifiers**

Let's understand the access modifiers by a simple table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

#### 2) Non-access Modifier

Non-access modifiers do not change the accessibility of variables and methods, but they do provide them special properties. Non-access modifiers are of 5 types,

1. Final
2. Static
3. Transient
4. Synchronized
5. Volatile

#### Final

Final modifier is used to declare a field as final i.e. it prevents its content from being modified. Final field must be initialized when it is declared.

*Example :*

class Cloth

{

final int MAX\_PRICE = 999; //final variable

final int MIN\_PRICE = 699;

final void display() //final method

{

System.out.println("Maxprice is" + MAX\_PRICE );

System.out.println("Minprice is" + MIN\_PRICE);

}

}

A class can also be declared as final. A class declared as final cannot be inherited. **String** class in java.lang package is a example of final class. Method declared as final can be inherited but you cannot override(redefine) it.

#### Static Modifier

Static Modifiers are used to create class variable and class methods which can be accessed without instance of a class. Lets study how it works with variables and member functions.

#### Static with Variables

Static variables are defined as a class member that can be accessed without any object of that class. Static variable has only one single storage. All the object of the class having static variable will have the same instance of static variable. Static variables are initialized only once.

Static variable are used to represent common property of a class. It saves memory. Suppose there are 100 employee in a company. All employee have its unique name and employee id but company name will be same all 100 employee. Here company name is the common property. So if you create a class to store employee detail, company\_name field will be mark as static.

**Example**

class Employee

{

int e\_id;

String name;

static String company\_name = "StudyTonight";

}

#### Example of static variable

class ST\_Employee

{

int eid;

String name;

static String company\_name ="StudyTonight";

public void show()

{

System.out.println(eid+" "+name+" "+company\_name);

}

public static void main( String[] args )

{

ST\_Employee se1 = new ST\_Employee();

se1.eid = 104;

se1.name = "Abhijit";

se1.show();

ST\_Employee se2 = new ST\_Employee();

se2.eid = 108;

se2.name = "ankit";

se2.show();

}

}

**Output**

104 Abhijit StudyTonight

108 ankit StudyTonight