Semester: IV						
OBJECT ORIENTED PROGRAMMING USING JAVA (Theory and Practice) (Common to CS and IS)						
Course Code	**:**	**18CS45**		**CIE Marks**	**:**	**100 + 50**
Credits: L:T:P	**:**	**3:0:1**		**SEE Marks**	**:**	**100 + 50**
Total Hours	**:**	**39L + 35P**		**SEE Duration**	**:**	**3 Hrs+3 Hrs**

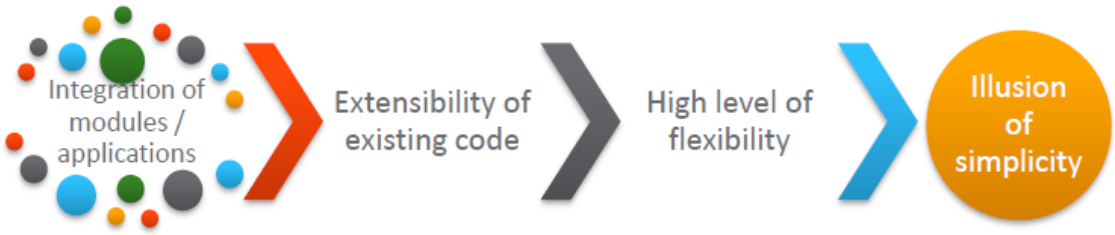
Unit 1

The Object Model

Foundations of the Object Model- Object-Oriented Programming , Object-Oriented Design, Object-Oriented Analysis , Elements of the Object Model - Abstraction , Encapsulation , Modularity , Hierarchy;

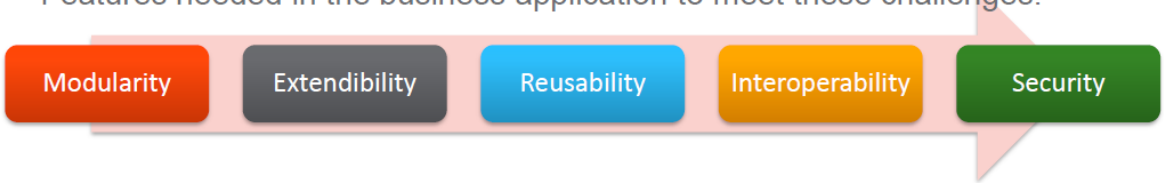
➢ Need for Object Oriented Approach

- Challenges in developing a business application

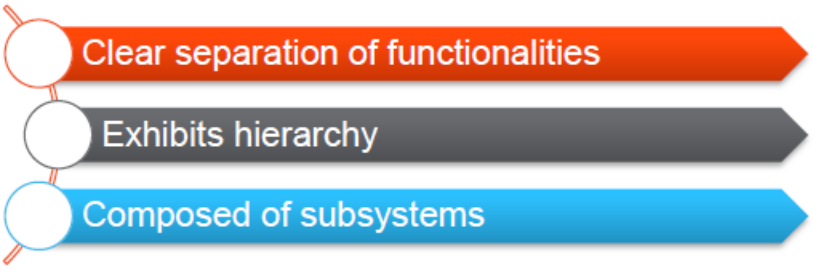


  - If these challenges are not addressed it may lead to software crisis

- Features needed in the business application to meet these challenges:



  - Challenges can be addressed through object oriented approach.
- Properties of a business application.

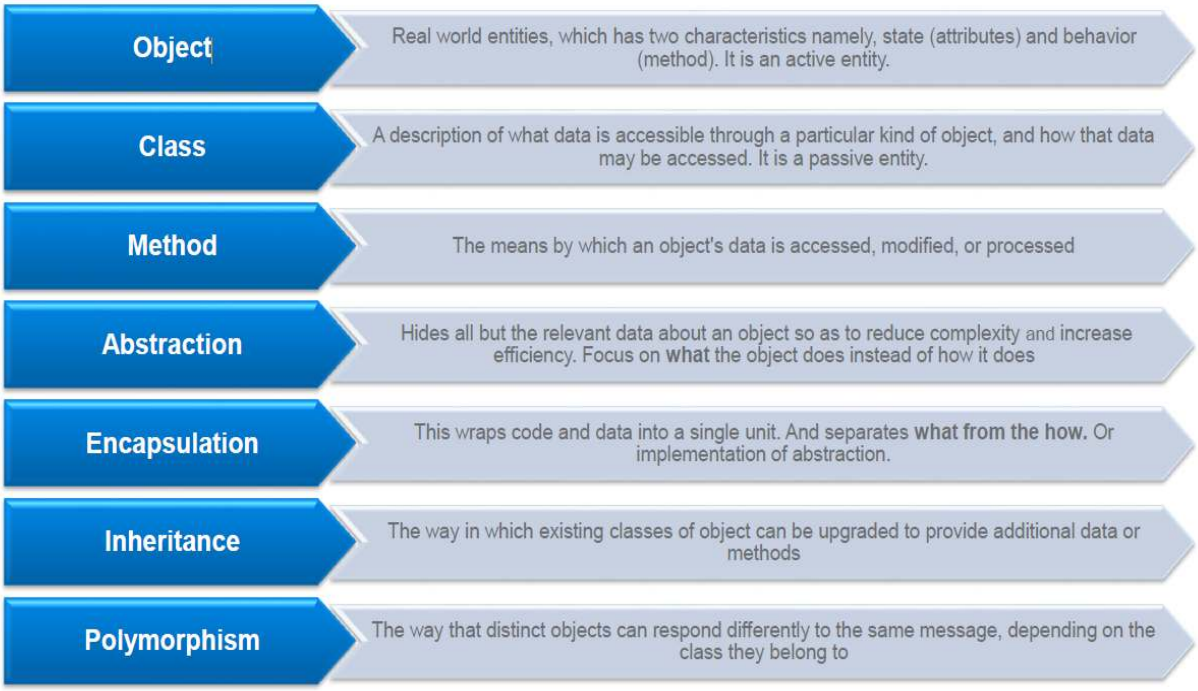


- These properties can be implemented using object oriented concepts.

➢ Object Oriented Programming

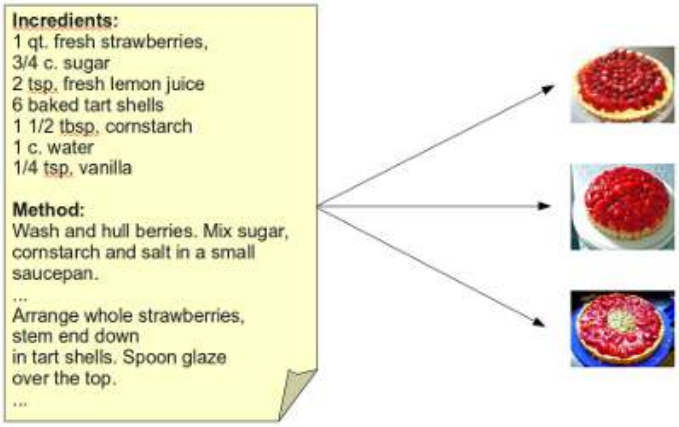
An object oriented program is based on classes and there exists a collection of interacting objects, as opposed to the conventional model, in which a program consists of functions and routines. In OOP, each object can receive messages, process data, and send messages to other objects.

OO Terminologies:



➢ Classes and Objects

Analogy: The Cake Class

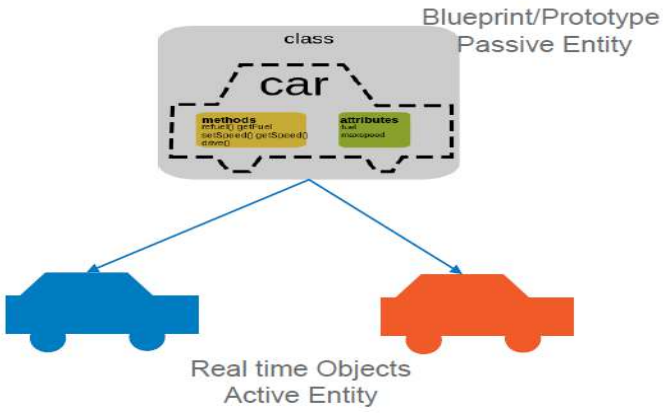


A class definition can be compared to the recipe to bake a cake. A recipe is needed to bake a cake. The main difference between a recipe (class) and a cake (an instance or an object of this class) is obvious. A cake can be eaten when it is baked, but you can't eat a recipe, unless you like the taste of printed paper.

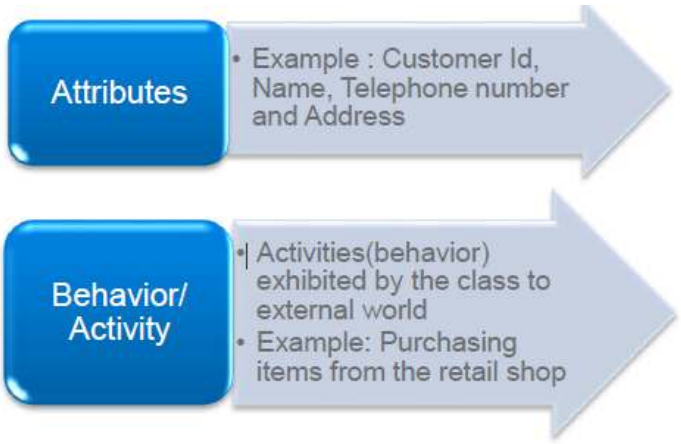
Like baking a cake, an OOP program constructs objects according to the class definitions of the program. A class contains variables and methods. If you bake a cake you need ingredients and instructions to bake the cake. Accordingly a class needs variables and methods. There are class variables, which have the same value in all methods and there are instance variables, which have normally different values for different objects.

A class also has to define all the necessary methods, which are needed to access the data.

A class is a prototype/ design that describes the common attributes (properties) and activities (behaviors) of objects.



Example 1: Consider a retail shop application – automating process of purchase by customers and billing, customers of the shop can be modelled as below:



Example 2:

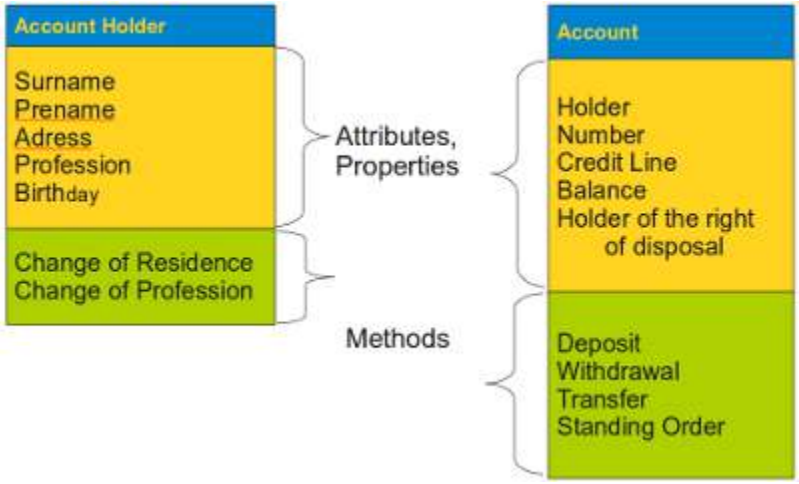
A class defines a data type, which contains variables, properties and methods. A class describes the abstract characteristics of a real-life thing - With the expression "real-life" thing here concepts (classes) like "bank account" or "account holder" to be considered. The abstract characteristics of a "thing" include its attributes and properties and the thing's behaviour, i.e. the methods and operations of this thing.

There can be instances and objects of classes. An instance is an object of a class created at run-time. In programmer vernacular, a strawberry tart is an instance of the strawberry recipe class. The set of values of the attributes of a particular object is called its state.

The object consists of state and the behaviour that's defined in the object's classes The terms object and instance are normally used synonymously.

As discussed, classes usually contain attributes and properties and methods for these instances and properties. Essentially, a method is a function, but it's a special kind of function which belongs to a class, i.e. it is defined within a class, and works on the instance and class data of this class. Methods can only be called through instances of a class or a subclass, i.e. the class name followed by a dot and the method name.

In the illustration below shows an example of two classes "Account" and "Account Holder". The data of the "Account Holder" consists, for example, of the Holder Surname and Prename, Adress, Profession, and Birthday. Methods are "Change of Residence" and "Change of Profession". This model is not complete, because we need more data and above all more methods like e.g. setting and getting the birthday of an account holder.



The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Class: A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

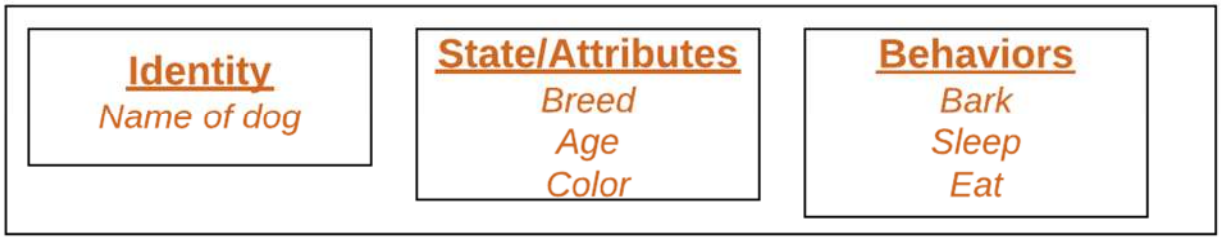
1. Modifiers: A class can be public or has default access.
2. Class name: The name should begin with a initial letter (capitalized by convention).

3. Superclass(if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. Interfaces(if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. Body: The class body surrounded by braces, { }.

Object: It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

1. State : It is represented by attributes of an object. It also reflects the properties of an object.
2. Behavior : It is represented by methods of an object. It also reflects the response of an object with other objects.
3. Identity : It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog



Method: A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping the code. In Java, every method must be part of some class which is different from languages like C, C++ and Python.

Methods are time savers and help us to reuse the code without retyping the code.

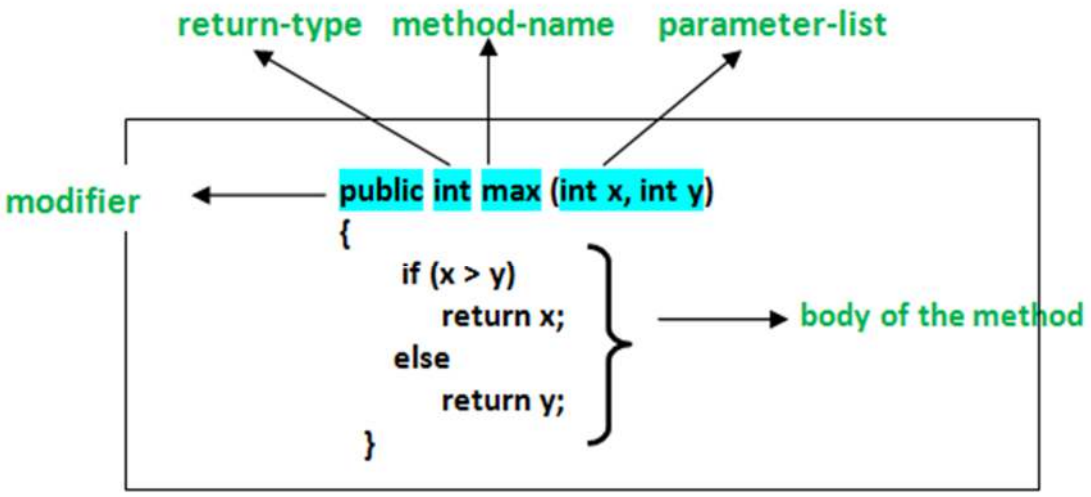
Method Declaration

In general, method declarations has six components:

6. Access Modifier: Defines access type of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
    - public: accessible in all class in your application.
    - protected: accessible within the package in which it is defined and in its subclass(es)(including subclasses declared outside the package)

- private: accessible only within the class in which it is defined.
- default (declared/defined without using any modifier): accessible within same class and package within which its class is defined.

7. The return type: The data type of the value returned by the method or void if does not return a value.
8. Method Name: the rules for field names apply to method names as well, but the convention is a little different.
9. Parameter list: Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
10. Exception list: The exceptions you expect by the method can throw, you can specify these exception(s).
11. Method body: it is enclosed between braces. The code you need to be executed to perform your intended operations.



Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Object Oriented approach benefits:

- ***Leads to development of smaller bu stable subsystems.***
- ***The subsystems are resilient to change.***
- ***Reduces the risk factor in building large systems as they are built incrementally from subsystems which are stable.***

Hence object oriented approach is suitable for developing extremely complex business systems.

➢ **Object Oriented System Development:**

- Object Orienetd Analysis and Design involve:
  - Modeling the system based on requirements.
  - Identification of classes and relationship between them.
  - Identification of attributes and methods.
  - Constructing the logical and physical object models.
- The representation of design is done usimg UML

Elements of the Object Model

- **Abstraction:** ***Process of identifying the essential details to be known and ignoring the non- essential details from the perspective of the user of the system.***
  Data Abstraction is the property by virtue of which only the essential details are displayed to the user.The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

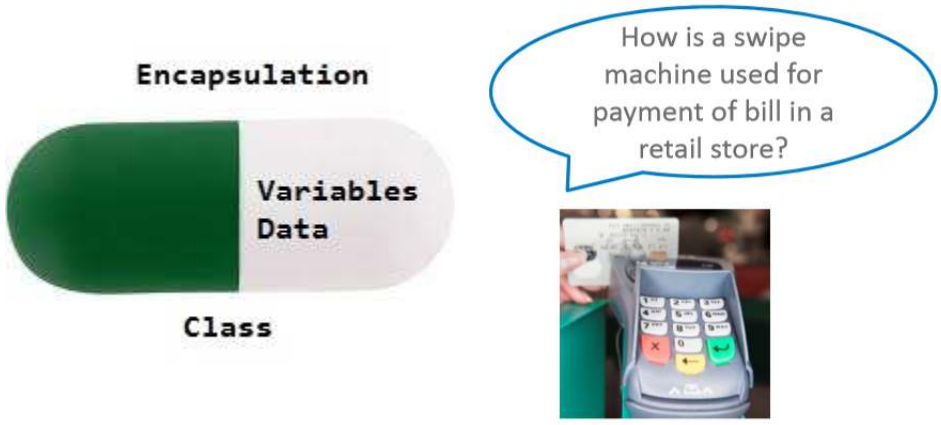
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

- ***Encapsulation: A mechanism of hiding the internal details and allowing a simple interface which ensures that the object can be used without having to know how it works.***

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
- Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.



- ***Inheritance: Inhertiance is a mechanism which allows to define generalized characteristics and behaviour and also create specialized ones. The specialized ones automatically tend to inherit all the properties of the generic ones.***

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.
Important terminology:

- **Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

- ***Polymorphism***: ***Refers to the ability of an object/ operation to behave differently in different situations.***
  Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently.

- Payment of bill - Two modes
  - Cash (Calculation includes VAT)

What do you observe in this retail store scenario?

Total Amount = Purchase amount + VAT

  - Credit card(Calculation includes processing charge and VAT)

Total Amount = Purchase amount + VAT + Processing charge

References:

1. Geeks for Geeks https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/

2. Infosys Foundation Programme materials

Java - Introduction

Java is a platform-independent programming language used to create secure and robust application that may run on a single computer or may be distributed among servers and clients over a network.

Java features such as platform-independency and portability ensure that while developing Java EE enterprise applications, you do not face the problems related to hardware , network , and the operating system.

History of Java

Java was started as a project called "Oak" by James Gosling in June 1991. Gosling's goals were to implement a virtual machine and a language that had a familiar C like notation but with greater uniformity and simplicity than C/C++.

The First publication of Java 1.0 was released by Sun Microsystems in 1995. It made the promise of "Write Once, Run Anywhere", with free runtimes on popular platforms.

In 2006-2007 Sun released java as open source and and plateform independent software.

Over time new enhanced versions of Java have been released. The current version of Java is Java 1.7 which is also known as Java 7.

Features of Java

The characteristics and features of java are as follows.

1) Simple

Java is a simple language because of its various features, Java Doesn?t Support Pointers , Operator Overloading etc. It doesn?t require unreferenced object because java support automatic garbage collection.

Java provides bug free system due to the strong memory management.

2) Object-Oriented

Object-Oriented Programming Language (OOPs) is the methodology which provide software development and maintenance by using object state, behavior , and properties.

Object Oriented Programming Language must have the following characteristics.

1)Encapsulation 2)Polymorphism 3)Inheritance 4)Abstraction

As the languages like Objective C, C++ fulfills the above four characteristics yet they are not fully object oriented languages because they are structured as well as object oriented languages.

In java everything is an Object. Java can be easily extended since it is based on the Object model

3) Secure

Java is Secure Language because of its many features it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption. Java does not support pointer explicitly for the memory.

- All Program Run under the sandbox.

4) Robust

Java was created as a strongly typed language. Data type issues and problems are resolved at compile-time, and implicit casts of a variable from one type to another are not allowed.

Memory management has been simplified java in two ways. First Java does not support direct pointer manipulation or arithmetic. This make it possible for a java program to overwrite memory or corrupt data.

Second , Java uses runtime garbage collection instead of instead of freeing of memory. In languages like c++, it Is necessary to delete or free memory once the program has finished with it.

5) Platform-independent.

Java Language is platform-independent due to its hardware and software environment. 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 byte code. This byte code is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

6) Architecture neutral

It is not easy to write an application that can be used on Windows , UNIX and a Macintosh. And its getting more complicated with the move of windows to non Intel CPU architectures.

Java takes a different approach. Because the Java compiler creates byte code instructions that are subsequently interpreted by the java interpreter, architecture neutrality is achieved in the implementation of the java interpreter for each new architecture.

7) Portable

Java code is portable. It was an important design goal of Java that it be portable so that as new architectures(due to hardware, operating system, or both) are developed, the java environment could be ported to them.

In java, all primitive types(integers, longs, floats, doubles, and so on) are of defined sizes, regardless of the machine or operating system on which the program is run. This is in direct contrast to languages like C and C++ that leave the sized of primitive types up to the compiler and developer.

Additionally, Java is portable because the compiler itself is written in Java.

8) Dynamic

Because it is interpreted , Java is an extremely dynamic language, At runtime, the java environment can extends itself by linking in classes that may be located on remote servers on a network(for example, the internet)

At runtime, the java interpreter performs name resolution while linking in the necessary classes. The Java interpreter is also responsible for determining the placement of object in memory. These two features of the Java interpreter solve the problem of changing the definition of a class used by other classes.

9) Interpreted

We all know that Java is an interpreted language as well. With an interpreted language such as Java, programs run directly from the source code.

The interpreter program reads the source code and translates it on the fly into computations. Thus, Java as an interpreted language depends on an interpreter program.

The versatility of being platform independent makes Java to outshine from other languages. The source code to be written and distributed is platform independent.

Another advantage of Java as an interpreted language is its error debugging quality. Due to this any error occurring in the program gets traced. This is how it is different to work with Java.

10) High performance

For all but the simplest or most infrequently used applications, performance is always a consideration for most applications, including graphics-intensive ones such as are commonly found on the world wide web, the performance of java is more than adequate.

11) Multithreaded

Writing a computer program that only does a single thing at a time is an artificial constraint that we?ve lived with in most programming languages. With java, we no longer have to live with this limitation. Support for multiple, synchronized threads is built directly into the Java language and runtime environment.

Synchronized threads are extremely useful in creating distributed, network-aware applications. Such as application may be communicating with a remote server in one thread while interacting with a user in a different thread.

12) Distributed.

Java facilitates the building of distributed application by a collection of classes for use in networked applications. By using java?s URL (Uniform Resource Locator) class, an application can easily access a remote server. Classes also are provided for establishing socket-level connections.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

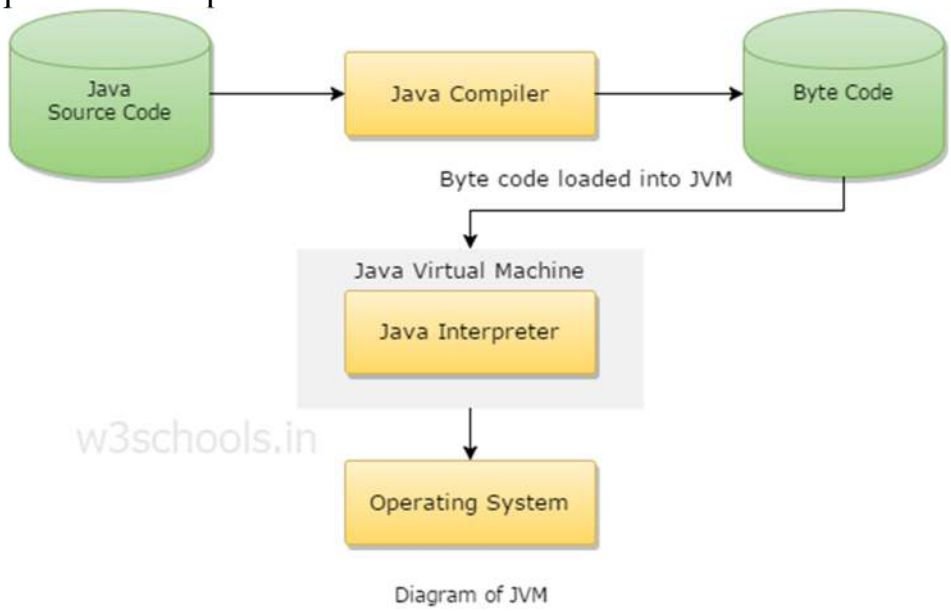


Diagram of JVM

JVM generates a **.class(Bytecode)** file, and that file can be run in any OS, but JVM should have in OS, because JVM is platform dependent.

The JVM performs following main tasks:

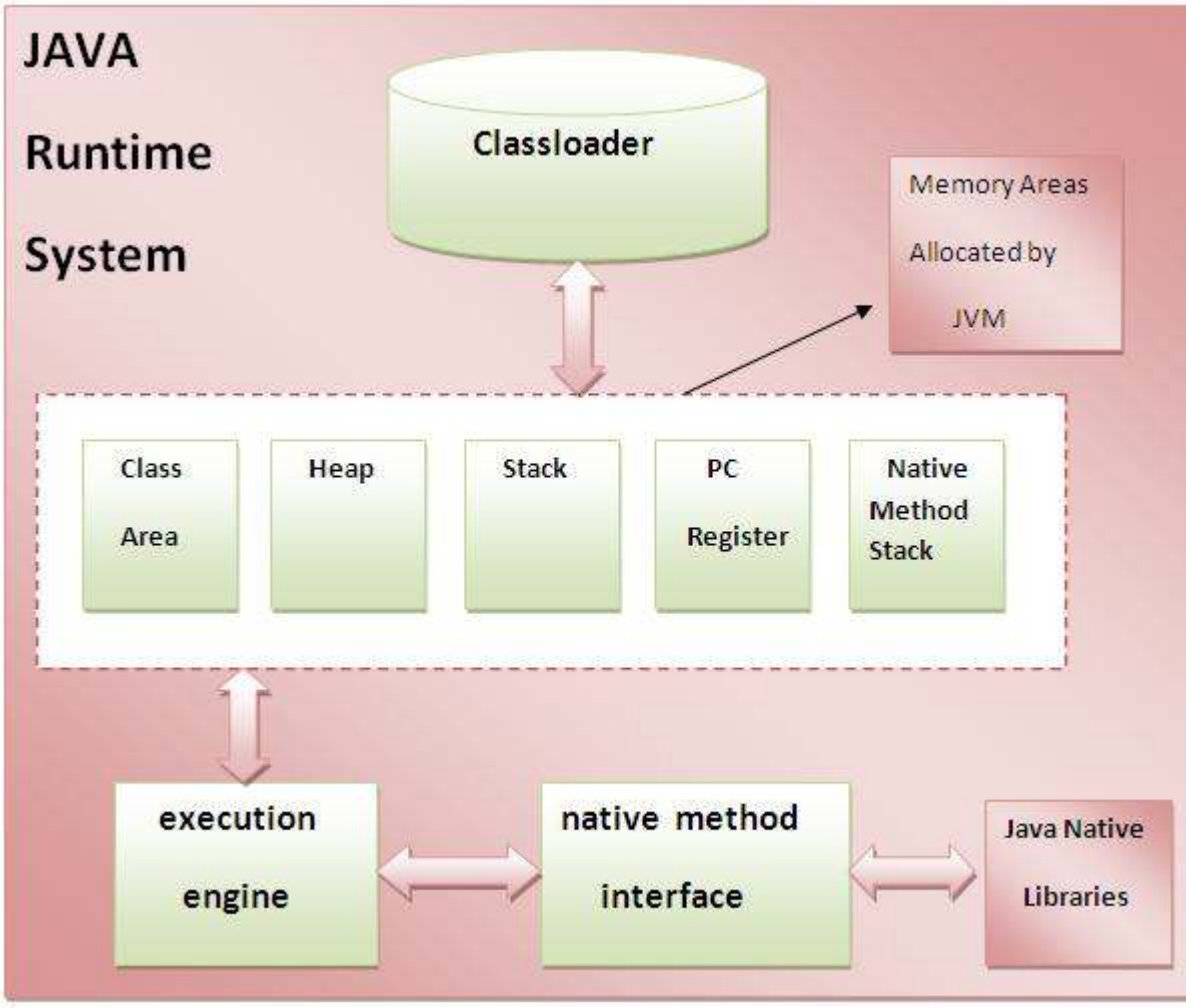
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Internal Architecture of JVM

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



1) Classloader:

Classloader is a subsystem of JVM that is used to load class files.

2) Class(Method) Area:

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap:

It is the runtime data area in which objects are allocated.

4) Stack:

Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register:

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack:

It contains all the native methods used in the application.

7) Execution Engine:

It contains:

1) A virtual processor

2) Interpreter: Read bytecode stream then execute the instructions.

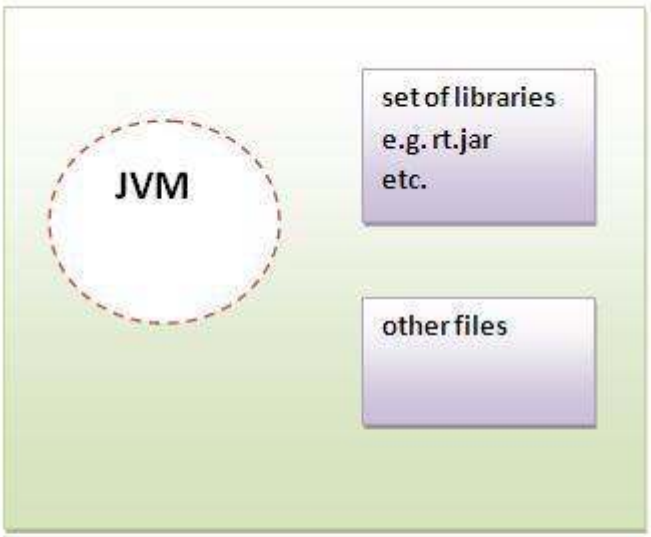
3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term compiler refers to a translator from the instruction set of a Java virtual machine

(JVM) to the instruction set of a specific CPU.

JRE

JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM.It physically exists.It contains set of libraries + other files that JVM uses at runtime.

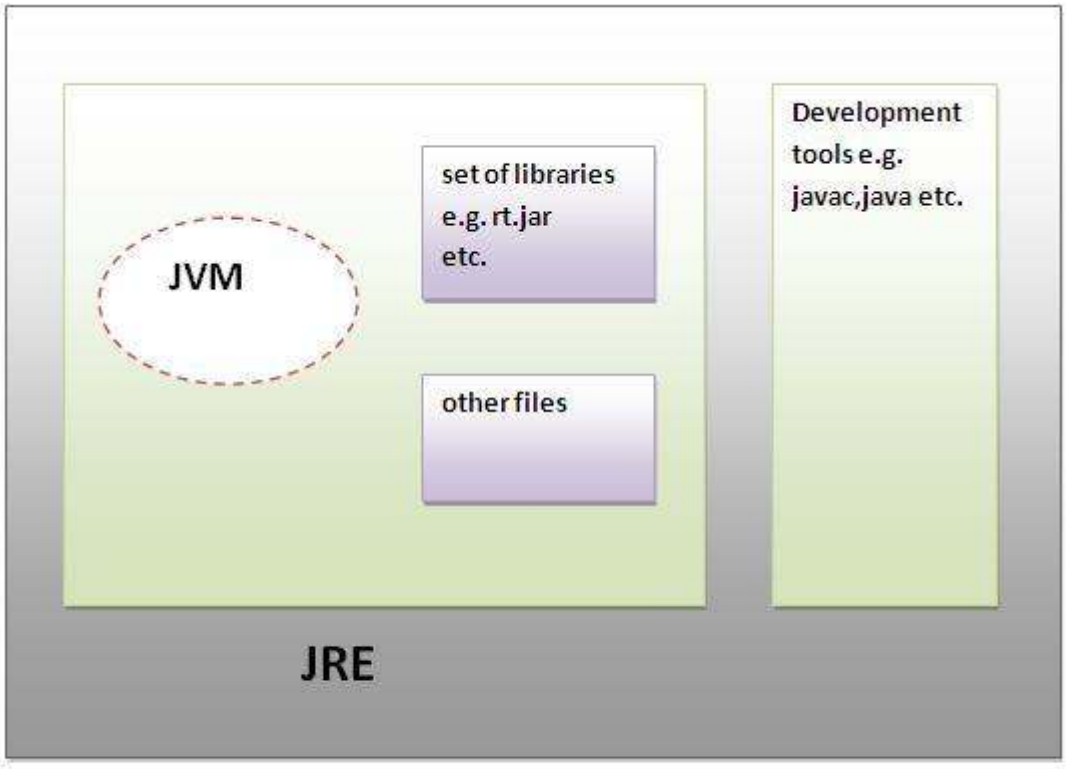
Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



JRE

JDK

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.



Basic JDK Tools

These tools are the foundation of the Java Development Kit.

javac

javac is the compiler for the Java programming language, it's used to compile *.java* file. It creates a *class* file which can be run by using java command.

Example:

```
c:javac TestFile.java
```

java

When a class file is created, the *java* command can be used to run the Java program.

Example:

```
c:java TestFile.class
```

Both run using command prompt. ***.java*** is the extension for java source files which are simple text files. After coding and saving it, the javac compiler is invoked for creating ***.class*** files. As the ***.class*** files get created, the Java command can be used to run java program.

javadoc

JavaDoc is a API documentation generator for the Java language, which generates documentation in HTML format from Java source code.

appletviewer

appletviewer run and debug applets without a web browser, its standalone command-line program to run Java applets.

jar

jar is (manage Java archive) a package file format that contains class, text, images and sound files for a Java application or applet gathered into a single compressed file.

Java - Basic Program

When we consider a Java program it can be defined as a collection of objects that communicate via invoking each others methods. Let us now briefly look into what do class, object, methods and instance variables mean.

- **Object -** Objects have states and behaviors. Example: A dog has states-color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.
- **Class -** A class can be defined as a template/ blue print that describe the behaviors/states that object of its type support.
- **Methods -** A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables -** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables

First Java Program:

Let us look at a simple code that would print the words *Simple Test Promgram*.

```
public class SimpleProgram {

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

```
        System.out.println("Simple Test Program");
    }
}
```

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

- Open notepad and add the code as above.
- Save the file as : SimpleProgram.java.
- Open a command prompt window and go to the directory where you saved the class. Assume its D:\.
- Type ' javac SimpleProgram.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).
- Now type ' java SimpleProgram' to run your program.
- You will be able to see ' Simple Test Program ' printed on the window.

```
C : > javac SimpleProgram.java
C : > java SimpleProgram
Simple Test Program
```

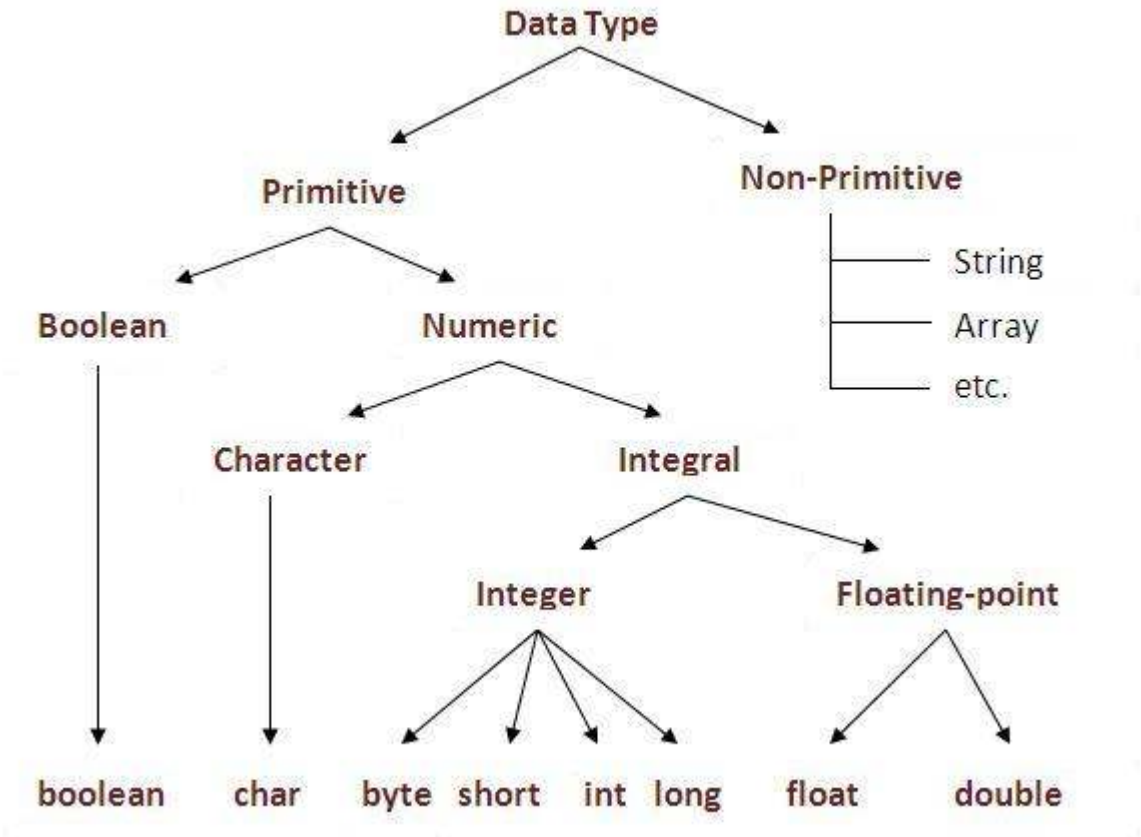
Explain public static void main (String args[])…..

The **public** keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared.In this case, main( ) must be declared as public, since it must be called by code outside of its class when the program is started. The keyword **static** allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since main( ) is called by the Java interpreter before any objects are made. The keyword **void** simply tells the compiler that main( ) does not return a value. As you will see, methods may also return values.
As stated, **main**( ) is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, Main is different from main. It is important to understand that the Java compiler will compile classes that do not contain a main( ) method. But the Java interpreter has no way to run these classes. So, if you had typed Main instead of main, the compiler would still compile your program. However, the Java interpreter would report an error because it would be unable to find the main( ) method.
Any information that you need to pass to a method is received by variables specified within the set of parentheses that follow the name of the method. These variables are called parameters. If there are no parameters required for a given method, you still need to include the empty parentheses. In main( ), there is only one parameter, albeit a complicated one. **String args[ ]** declares a parameter named args, which is an array of instances of the class String. Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

Data Types in Java

In java, there are two types of data types

- primitive data types
- non-primitive data types



Data Type Default Value Default size

boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte

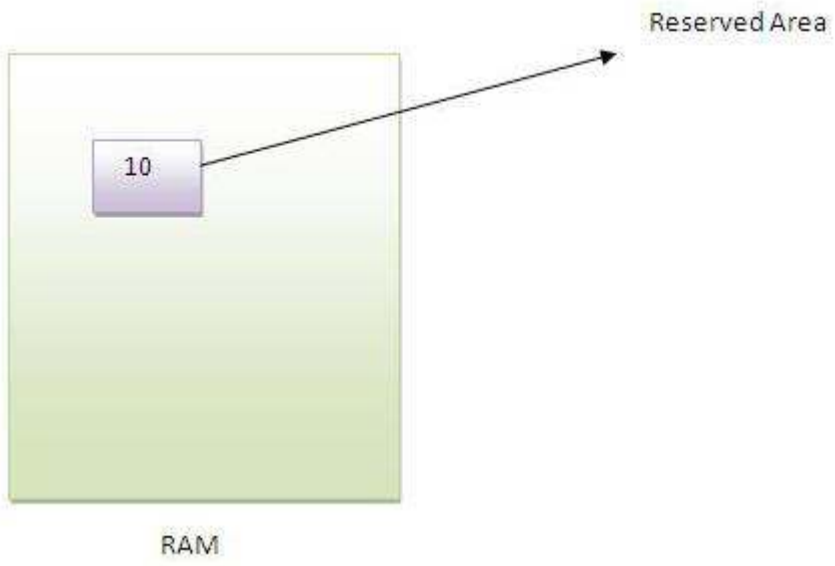
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.
lowest value:\u0000
highest value:\uFFFF

Variable

Variable is name of reserved area allocated in memory.



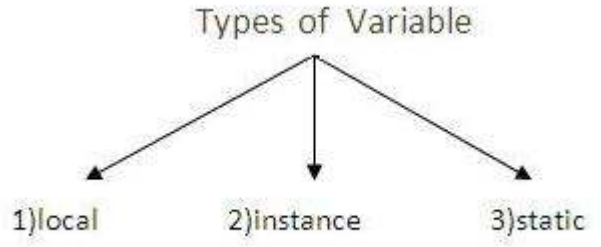
1. int data=50;//Here data is variable

Types of Variable

There are three types of variables in java

- local variable

- instance variable
- static variable



Local Variable

A variable that is declared inside the method is called local variable.

Instance Variable

A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

Static variable

A variable that is declared as static is called static variable. It cannot be local.

Example to understand the types of variables

```
1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class
```

Java - Arrays

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

The first element of array start with zero.

Using an array in your program is a 3 step process:

1. Declaring you array.
2. Constructing your array.
3. Initializing your array.

Declaring Array:-

Syntax:

```
elementType[] arrayName;
Or
elementType arrayName[];
```

Example:

```
int[] intArray;
int intArray[];
```

Constructing Array:-

Syntax:

```
new  elementType[size];
```

Example:

```
int[] intArray = new int[10]; // Defines that intArray will store 10 integer
values
int intArray[] = new int[10];
```

Initializing Array:-

Syntax:

```
arrayName[element 0,1,2?.. N] = value;
```

Example:

```
intArray[0] = 10; // Assign an integer value 10 to the first element 0 of the
array
intArray[1] = 20;
```

Declaring and Initializing Array:-

Syntax:

```
elementType[] arrayName = {values1,values2,? valueN};
```

Example:

```
int[] intArray = {1,2,3,4};
```

Array Example:

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

    String[] names = new String[3];
    names[0] = "A";
```

```
    names[1] = "ABC";
    names[2] = "XYZ";
    for (int i = 0; i < names.length; i++) {
      System.out.println(names[i]);
    }
    //this line should throw an exception
    //System.out.println(names[6]);
  }
}
```

Array are passed by refrence:

Arrays are passed to functions by reference, or as a pointer to the original. This means anything you do to the Array inside the function affects the original.

Example:

```
public class Main{
        public static void passByRefrence(String a[]){
                a[0] = "Z";
        }
        public static void main(String args[]){
                String[] b = {"A","B","C"};
                System.out.println("Before Function call  :  "+b[0]);
                Main.passByRefrence(b);
                System.out.println("After Function call : "+b[0]);
        }
}
```

Output:

Before Function call : A

After Function call : Z

Multidimensional Arrays:

Multidimensional arrays, are arrays of arrays.

Syntax:

```
elementType[][] arrayName = new elementType[size][size];
```

Example:

```
int[][] intArrays = new int[4][5];
```

When you allocate memory for a multidimensional array, you need only specify the memory for the first (leftmost) dimension.

You can allocate the remaining dimensions separately.

In Java the length of each array in a multidimensional array is under your control.

Example:

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

Array of Objects:

It is possible to create array of objects of user created class.

Example:

```
class Employee{
        int id;
        String name;
        public void setData(int id, String name){
                this.id = id;
                this.name = name;
        }
        public void displayData(){
                System.out.println("Employee ID : "+this.id);
                System.out.println("Employee Name : "+this.name);
        }
}
class Main{
        public static void main(String args[]){
                Employee[] emp = new Employee[2];
                emp[0].setData(1,"ABC");
                emp[1].setData(2,"XYZ");
                emp[0].displayData();
                emp[1].displayData();
```

```
        }
}
```

Output:

Employee ID : 1

Employee Name : ABC

Employee ID : 2

Employee Name : XYZ

Operators in java

Operator in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr*~ !
multiplicative	* / %
additive	+ -
shift	<<>>>>
relational	<><= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^

bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java - Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators

Arithmetic Operators

Arithmetic operators are used in mathematical like addition, subtraction etc. The following table lists the arithmetic operators:

Assume that int X = 10 and int Y = 20

Operators	Description
+	Addition – Adds values on either side of the operator
-	Subtraction – Subtracts right hand operand from left hand operand
*	Multiplication – Multiplies values on either side of the operand
/	Division - Divides left hand operand by right hand operand
%	Modulus - Divides left hand operand by right hand operand and returns remainder
++	Increment - Increase the value of operand by 1
--	Decrement - Decrease the value of operand by 1

Example:

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

```
                Int X = 10;
                Int Y = 20;
                System.out.println("Addition (X+Y) = "+(X+Y)); // return 30
                System.out.println("Subtraction  (X-Y) = "+(X-Y)); // return -
10
                System.out.println("Multiplication (X*Y) = "+(X*Y)); // return
200
                System.out.println("Division (Y/X) = "+(Y/X)); // return 2
                System.out.println("Addition (Y%X) = "+(Y%X)); // return 0
                Y++;
                System.out.println("Increment Y = "+Y); // return 21
                X--;
                System.out.println("Decrement X = "+X); // return 9
        }
}
```

Relational Operators

There are following relational operators supported by Java language like ==, ! = etc.

Assume variable X=10 and variable Y=20 then:

Operator	Description
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Example :

```
public class Main{
                public static void main(String args[]){
                int X = 10;
                int Y = 20;
                System.out.println("(X == Y) = "+(X == Y));
                System.out.println("(X != Y) = "+(X != Y));
```

```
                System.out.println("(X > Y) = "+(X > Y));
                System.out.println("(X < Y) = "+(X < Y));
                System.out.println("(X >= Y) = "+(X >= Y));
                System.out.println("(X <= Y) = "+(X <= Y));
        }
}
```

Bitwise Operators

Java defines several bitwise operators like &, | etc which can be applied to the integer types(long, int, short, char, and byte).

Bitwise operator works on bits(0 or 1) and perform bit by bit operation. Assume if x = 60; and y = 13; Now in binary format they will be as follows:

x = 0011 1100

y = 0000 1101

---

x&y = 0000 1100

x|y = 0011 1101

x^y = 0011 0001

~x = 1100 0011

The following table lists the bitwise operators:

Assume integer variable X=60 and variable Y=13 then:

Operator	Description
&	Binary AND Operator copies a bit to the result if it exists in both operands.
|	Binary OR Operator copies a bit if it exists in eather operand.
^	Binary XOR Operator copies the bit if it is set in one operand but not both.
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.

Operator	Description
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.

Example :

```
public class Main{
        public static void main(String args[]){
                int X = 60;
                int Y = 13;
                System.out.println("(X & Y) = "+(X & Y));
                System.out.println("(X | Y) = "+(X | Y));
                System.out.println("(X ^ Y) = "+(X ^ Y));
                System.out.println("(~X) = "+(~X));
                System.out.println("(X << Y) = "+(X << 2));
                System.out.println("(X >> Y) = "+(X >> 3));
                System.out.println("(X >>> Y) = "+(X >>> 1));
        }
}
```

Logical Operators

The following table lists the logical operators like &&, || etc. This logical operator use for join two condition.

Assume boolean variables X=true and variable Y=false then:

Operator	Description
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.
!	Called Logical NOT Operator. Use to reverses the

Operator	Description
	logical state of its operand. If a condition is true then Logical NOT operator will make false.

Example:

```
public class Main{
        public static void main(String args[]){
                int X = 60;
                int Y = 13;
                if((X == Y) && (X != Y)){
                        System.out.println("True");
                }else{
                        System.out.println("False");
                }
                if((X == Y) || (X != Y)){
                        System.out.println("True");
                }
                else{
                        System.out.println("False");
                }
        }
}
```

Assignment Operators

There are following assignment operators supported by Java language:

Operator	Description
=	Simple assignment operator, Assigns values from right side operands to left side operand
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand

Operator	Description
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand
<<=	Left shift AND assignment operator
>>=	Right shift AND assignment operator
&=	Bitwise AND assignment operator
^=	Bitwise exclusive OR and assignment operator
|=	Bitwise inclusive OR and assignment operator

Example:

```
public class Main{
        public static void main(String args[]){
                int X = 60;
                int Y = 13;
                X += 1;
                System.out.println("X+=1 : "+X);
                Y<<=1;
                System.out.println("Y<<=1 : "+Y);
                /* Return 26 : 13(binary - 00001101) shift one bit left means
26(00011010) */
        }
}
```

Java - Loop control

Loop is very common control flow statement in programming languages such as java. We are going to describe the basics of “java loop”. In this post, we will learn various ways to use loop in day-to-day programming habits.

There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop

There are four types of loops:

1. For loop
2. For each loop
3. While loop
4. Do..While loop

For Loop

It is structured around a finite set of repetitions of code. So if you have a particular block of code that you want to have run over and over again a specific number of times the For Loop is helpful.

Syntax:

```
for(initilization; conditional expression; increment
expression)

{
   //repetition code here
}
```

Example:

```
public class Example {
      public static void main(String args[]) {
           for(int x = 50; x < 55; x++) {
                System.out.println("Value of x : " + x
);

           }
      }

}
```

Output:

Value of x : 50;

Value of x : 51;

Value of x : 52;

Value of x : 53;

Value of x : 54;

For each Loop

This loop is supported from Java 5.

For each loop is mainly used for iterate the Array, List etc.

Syntax:

```
for(declaration : expression)
{
    //Code Here
}
```

Example:

```
        public class Example {

   public static void main(String args[]){

      List list = new ArrayList();
      list.add(10);
      list.add(20);
      list.add(30);
      System.out.print("List Value = ");
      for(int x : list){
         System.out.print( x );
         System.out.print(",");
      }
      String [] names ={"abc", "xyz", "test", "example"};
      System.out.println("String Array value = ");
      for( String name : names ) {
         System.out.print( name );
         System.out.print(",");
      }
   }
}
```

Output:

List Value = 10,20,30

String Array value = abc,xyz,test,example

While Loop

Another looping strategy is known as the While Loop. The While Loop is good when you don't want to repeat your code a **specific** number of times, rather, you want to **keep looping** through your code until a **certain condition is met**.

Syntax:

```
while(Boolean_expression)
{
   //Repetition Code Here
}
```

Example:

```
public class Example {

   public static void main(String args[]) {
      int x = 50;

      while( x < 55 ) {
         System.out.println("Value of x : " + x );
         x++;
      }
   }
}
```

Output:

Value of x : 50

Value of x : 51

Value of x : 52

Value of x : 53

Value of x : 54

Do..While Loop

This type of loop is used in very rare cases because it does the same thing as a while loop does, except that a do..while loop is guaranteed to execute at least on time.

Syntax:

```
do
{
   //Code Here
}while(Boolean_expression);
```

Example:

```
public class Test {

   public static void main(String args[]){
      int x = 50;

      do{
         System.out.println("Value of x : " + x );
         x++;
      }while( x < 50 );
   }
}
```

Output:

Value of x : 50

Above example first execute code inside loop and then check condition that's why it's display 50.

Java - Decision Making

If statement:

The if statement is Java's conditional branch statement.

Syntax:

```
If(Boolean expression){
```

```
// Code here
}
```

Example:

```
public class Main{
        public static void main(String args[]){
                int a = 10;
                int b = 20;
                if(a>b){
                        System.out.println("a is greater than b");
                }
                if(b<a){
                        System.out.println("b is less than a");
                }
}
}
```

Output:

a is greater than b

b is less than a

if else :

Syntax:

```
if(Boolean condition){
        // statement1
}else{
        // statement2
}
```

This if else work like : if condition is true than statement1 is executed otherwise statement2 is executed.

Example:

```
public class Main{
```

```
        public static void main(String args[]){
                int a = 10;
                int b = 20;
                if(a>b){
                        System.out.println(?a is greater than b?);
                 }else{
                        System.out.println(?b is greater than a?);
                 }
        }
}
```

Output:

b is greater than a

If-else-if-ladder:

Syntax:

```
if(Boolean condition){
        // statement1
}else if(Boolean condition){
        // statement2
}else if(Boolean condition){
        // statement3
}
........
else{
        // else statement
}
```

The if statement executed from top to down. As soon as one of the condition is true, statement associated with that if statement executed.

If none of the condition is true than else statement will be executed, only one of the statement executed from list of else if statements.

Example:

```
public class Main{
        public static void main(String args[]){
                int percentage = 65;
                if(percentage >= 70){
                        System.out.println(?First class with Distinction?);
                }else If(percentage >= 60){
                        System.out.println(?First Class?);
                }else If(percentage >= 48){
                        System.out.println(?Second Class?);
                }else If(percentage >= 36){
                        System.out.println(?Pass Class?);
                }else{
                        System.out.println(?Fail?);
                 }

        }
}
```

Output:

First Class

switch statement:

The switch statement is multi way branch statement in java programming. It is use to replace multilevel if-else-if statement.

Syntax:

```
switch(expression){
case value 1:
        // statement 1
        break;
case value 2:
        // statement 2
        break;
case value n:
        // statement n
        break;
default:
        //statements
        break;
}
```

The expression type must be the byte, short, int and char.

Each case value must be a unique literal(constant not a variable). Duplicate case value not allowed.

The each case value compare with expression if match found than corresponding statement will be executed. If no match is found than default statement will be executed. Default case if optional.

The break statement use to terminate statement sequence, if break statement is not written than all statement execute after match statement.

Example:

```
public class Main{
	public static void main(String args[]){
		int day = 1;
		switch(day){
			case 0:
				System.out.println("Sunday");
				 break;
			case 1:
				System.out.println("Monday");
				break;
			case 3:
				System.out.println("Tuesday");
				break;
			case 4:
				System.out.println("Wednesday");
				break;
			case 5:
				System.out.println("Thursday");
				break;
			case 6:
				System.out.println("Friday");
				break;
			case 7:
				System.out.println("Saturday");
				break;
			default:
				System.out.println("Invalid Day");
				break;
		}
	}
```

```
}
```

Output:

Monday

Note : switch statement support string from Java 7, means use string object in the switch expression.

Java - Modifier Types

Modifiers are keywords that you add to those definitions to change their meanings. The Java language has a wide variety of modifiers, including the following:

- Java Access Modifiers
- Non Access Modifiers

Access Control Modifiers:

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

private

If a method or variable is marked as private, then only code inside the same class can access the variable, or call the method. Code inside subclasses cannot access the variable or method, nor can code from any external class.

If a class is marked as private then no external class an access the class. This doesn't really make so much sense for classes though. Therefore, the access modifier private is mostly used for fields, constructors and methods.

Example :

```
public class Clock {
```

```
    private long time = 0;

}
```

Mostly private access modifier use for fields and make getter, setter method to access these fields.

default

The default access level is declared by not writing any access modifier at all. Default access levels means that code inside the class itself + code inside classes in the same package as this class, can access the class, field, constructor or method. Therefore, the default access modifier is also sometimes called a package access modifier.

Subclasses cannot access methods and member variables in the superclass, if they have default accessibility declared, unless the subclass is located in the same package as the superclass.

Example:

```
public class Clock {
     long time = 0;
}

public class ClockReader {
    Clock clock = new Clock();
    public long readClock{
      return clock.time;
    }
}
```

protected

The protected acces modifier does the same as the default access, except subclasses can also access protected methods and member variables of the superclass. This is true even if the subclass is not located in the same package as the superclass.

Example:

```
public class Clock {
    protected long time = 0; // time in milliseconds
}
public class SmartClock() extends Clock {
```

```
    public long getTimeInSeconds() {
     return this.time / 1000;
    }
}
```

public

The public access modifier means that all code can access the class, field, constructor or method, regardless of where the accessing code is located.

Example:

```
public class Clock {
   public long time = 0;
}
public class ClockReader {
   Clock clock = new Clock();
   public long readClock{
      return clock.time;
   }
}
```

Non Access Modifiers:

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.

The static Modifier:

Static Variables:

The *static* key word is used to create variables that will exist independently of any instances created for the class. Only one copy of the static variable exists regardless of the number of instances of the class.

Static variables are also known as class variables. Local variables cannot be declared static.

Static Methods:

The static key word is used to create methods that will exist independently of any instances created for the class.

Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.

Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method.

Example:

The static modifier is used to create class methods and variables, as in the following example:

```
public class InstanceCounter {

   private static int numInstances = 0;

   protected static int getCount() {
      return numInstances;
   }

   private static void addInstance() {
      numInstances++;
   }

   InstanceCounter() {
      InstanceCounter.addInstance();
   }

   public static void main(String[] arguments) {
      System.out.println("Starting with " +
      InstanceCounter.getCount() + " instances");
      for (int i = 0; i < 500; ++i){
         new InstanceCounter();
          }
      System.out.println("Created " +
      InstanceCounter.getCount() + " instances");
   }
}
```

This would produce following result:

```
Started with 0 instances
Created 500 instances
```

The final Modifier:

final Variables:

A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to an different object.

However the data within the object can be changed. So the state of the object can be changed but not the reference.

With variables, the *final* modifier often is used with *static* to make the constant a class variable.

Example:

```
public class Test{
  final int value = 10;
  // The following are examples of declaring constants:
  public static final int BOXWIDTH = 6;
  static final String TITLE = "Manager";

  public void changeValue(){
     value = 12; //will give an error
  }
}
```

final Methods:

A final method cannot be overridden by any subclasses. As mentioned previously the final modifier prevents a method from being modified in a subclass.

The main intention of making a method final would be that the content of the method should not be changed by any outsider.

Example:

You declare methods using the *final* modifier in the class declaration, as in the following example:

```
public class Test{
    public final void changeName(){
       // body of method
    }
}
```

final Classes:

The main purpose of using a class being declared as *final* is to prevent the class from being subclassed. If a class is marked as final then no class can inherit any feature from the final class.

Example:

```
public final class Test {
   // body of class
}
```

The abstract Modifier:

abstract Class:

An abstract class can never be instantiated. If a class is declared as abstract then the sole purpose is for the class to be extended.

A class cannot be both abstract and final. (since a final class cannot be extended). If a class contains abstract methods then the class should be declared abstract. Otherwise a compile error will be thrown.

An abstract class may contain both abstract methods as well normal methods.

Example:

```
abstract class Caravan{
   private double price;
   private String model;
   private String year;
   public abstract void goFast(); //an abstract method
   public abstract void changeColor();
}
```

abstract Methods:

An abstract method is a method declared with out any implementation. The methods body(implementation) is provided by the subclass. Abstract methods can never be final or strict.

Any class that extends an abstract class must implement all the abstract methods of the super class unless the subclass is also an abstract class.

If a class contains one or more abstract methods then the class must be declared abstract. An abstract class does not need to contain abstract methods.

The abstract method ends with a semicolon. Example: public abstract sample();

Example:

```
public abstract class SuperClass{
    abstract void m(); //abstract method
}
```

```
class SubClass extends SuperClass{
     // implements the abstract method
      void m(){
          .........
      }
}
```

Java - Abstraction

Abstraction is process of **hiding the implementation details** and showing only the functionality.

Abstraction in java is achieved by using interface and abstract class. Interface give 100% abstraction and abstract class give 0-100% abstraction.

A class that is declared as **abstract** is known as abstract class.

Syntax:
abstract class <class-name>{}

An abstract class is something which is incomplete and you cannot create instance of abstract class.
If you want to use it you need to make it complete or concrete by extending it.

A class is called concrete if it does not contain any abstract method and implements all abstract method inherited from abstract class or interface it has implemented or extended.

Abstract method in Java

A method that is declare as abstract and **does not have implementation** is known as abstract method.
If you define abstract method than class must be abstract.

Syntax:

abstract return_type method_name ();

An abstract method in Java doesn't have body, it's just a declaration. In order to use abstract method you need to **override** that method in Subclass.

Example 1 :(Without abstract method)

```
class Employee extends Person {

    private String empCode;
```

```
    public String getEmpCode() {
        return empCode;
    }

    public void setEmpCode(String empCode) {
        this.empCode = empCode;
    }


}

abstract  class Person {

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }



}

public class Main{
        public static void main(String args[]){
        //INSTIATING AN ABSTRACT CLASS GIVES COMPILE TIME ERROR
        //Person p =  new Person() ;


        //THIS REFERENCE VARIABLE CAN ACESS ONLY THOSE METHOD WHICH ARE
OVERRIDDEN
        Person person = new Employee();
        person.setName("Jatin Kansagara");
        System.out.println(person.getName());
    }
}
```

Example 2: (with abstract method)

```
public class Main{
        public static void main(String args[]){
        TwoWheeler test = new Honda();
        test.run();
    }
}
abstract  class TwoWheeler {
    public abstract void run();
}
class Honda extends TwoWheeler{
        public void run(){
```

```
                System.out.println("Running..");
        }
}
```

Java - Encapsulation

Encapsulation is process of **wrapping code** and data together into a single unit.

We can create fully encapsulated class by making the entire **data member will be private** and create getter, setter method to access that data member.

Encapsulation is also known as "**Data hiding**" because they are protecting data which is prone to change.

Example:

```
class Employee{
        private int employeeId;

        private String employeeName;

        private String designation;

        public int getEmployeeId() {
                return employeeId;
        }
        public void setEmployeeId(int employeeId) {
                this.employeeId = employeeId;
        }
        public String getEmployeeName() {
                return employeeName;
        }
        public void setEmployeeName(String employeeName) {
                this.employeeName = employeeName;
        }
        public String getDesignation() {
                return designation;
        }
        public void setDesignation(String designation) {
                this.designation = designation;
        }
}
```

Here are some **advantages** to use encapsulation in Java Code.

- Encapsulated Code is more flexible and easy to change with new requirements.
- By providing only getter and setter method access, you can make the class read only.
- Encapsulation in Java makes unit testing easy.
- A class can have total control over what is stored in its fields. Suppose you want to set the value of marks field i.e. marks should be positive value, than you can write the logic of positive value in setter method.
- Encapsulation also helps to write immutable class in Java which are a good choice in multi-threading environments.
- Encapsulation allows you to change one part of code without affecting other part of code.

The synchronized Modifier:

The synchronized key word used to indicate that a method can be accessed by only one thread at a time. The synchronized modifier can be applied with any of the four access level modifiers.

Example:

```
public synchronized void showDetails(){
.......
}
```

The transient Modifier:

An instance variable is marked transient to indicate the JVM to skip the particular variable when serializing the object containing it.

This modifier is included in the statement that creates the variable, preceding the class or data type of the variable.

Example:

```
public transient int limit = 55;   // will not persist
public int b; // will persist
```

The volatile Modifier:

The volatile is used to let the JVM know that a thread accessing the variable must always merge its own private copy of the variable with the master copy in the memory.

Accessing a volatile variable synchronizes all the cached copied of the variables in the main memory. Volatile can only be applied to instance variables which are of type object or private. A volatile object reference can be null.

Example:

```
public class MyRunnable implements Runnable
{
    private volatile boolean active;

    public void run()
    {
        active = true;
        while (active) // line 1
        {
            // some code here
        }
    }
    public void stop()
    {
        active = false; // line 2
    }
}
```

Usually, run() is called in one thread (the one you start using the Runnable), and stop() is called from another thread. If in line 1 the cached value of active is used, the loop may not stop when you set active to false in line 2. That's when you want to use *volatile*.

Java - Methods

Previous Page

Next Page

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.**println()** method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

Creating Method

Considering the following example to explain the syntax of a method −

Syntax

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

- **public static** − modifier
- **int** − return type
- **methodName** − name of the method
- **a, b** − formal parameters
- **int a, int b** − list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- **modifier** − It defines the access type of the method and it is optional to use.
- **returnType** − Method may return a value.
- **nameOfMethod** − This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List** − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body** − The method body defines what the method does with the statements.

Example

Here is the source code of the above defined method called **min()**. This method takes two parameters num1 and num2 and returns the maximum between the two −

```
/** the snippet returns the minimum between two numbers */

public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Method Calling

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when −

- the return statement is executed.
- it reaches the method ending closing brace.

The methods returning void is considered as call to a statement. Lets consider an example −

```
System.out.println("This is tutorialspoint.com!");
```

The method returning value can be understood by the following example −

```
int result = sum(6, 9);
```

Following is the example to demonstrate how to define a method and how to call it −

Example

Live Demo

```
public class ExampleMinNumber {

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("Minimum Value = " + c);
   }

   /** returns the minimum of two numbers */
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }
}
```

This will produce the following result −

Output

```
Minimum value = 6
```

The void Keyword

The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering a void method *methodRankPoints*. This method is a void method, which does not return any value. Call to a void method must be a statement i.e. *methodRankPoints(255.7);*. It is a Java statement which ends with a semicolon as shown in the following example.

Example

Live Demo

```
public class ExampleVoid {

   public static void main(String[] args) {
      methodRankPoints(255.7);
   }

   public static void methodRankPoints(double points) {
      if (points >= 202.5) {
         System.out.println("Rank:A1");
      }else if (points >= 122.4) {
         System.out.println("Rank:A2");
      }else {
         System.out.println("Rank:A3");
      }
   }
}
```

This will produce the following result −

Output

```
Rank:A1
```

Passing Parameters by Value

While working under calling process, arguments is to be passed. These should be in the same order as their respective parameters in the method specification. Parameters can be passed by value or by reference.

Passing Parameters by Value means calling a method with a parameter. Through this, the argument value is passed to the parameter.

Example

The following program shows an example of passing parameter by value. The values of the arguments remains the same even after the method invocation.

Live Demo

```
public class swappingExample {

   public static void main(String[] args) {
      int a = 30;
      int b = 45;
      System.out.println("Before swapping, a = " + a + " and b = " + b);

      // Invoke the swap method
      swapFunction(a, b);
      System.out.println("\n**Now, Before and After swapping values will be
same here**:");
      System.out.println("After swapping, a = " + a + " and b is " + b);
   }

   public static void swapFunction(int a, int b) {
      System.out.println("Before swapping(Inside), a = " + a + " b = " + b);

      // Swap n1 with n2
      int c = a;
      a = b;
      b = c;
      System.out.println("After swapping(Inside), a = " + a + " b = " + b);
   }
}
```

This will produce the following result −

Output

```
Before swapping, a = 30 and b = 45
Before swapping(Inside), a = 30 b = 45
After swapping(Inside), a = 45 b = 30

**Now, Before and After swapping values will be same here**:
After swapping, a = 30 and b is 45
```

Method Overloading

When a class has two or more methods by the same name but different parameters, it is known as method overloading. It is different from overriding. In overriding, a method has the same method name, type, number of parameters, etc.

Let's consider the example discussed earlier for finding minimum numbers of integer type. If, let's say we want to find the minimum number of double type. Then the concept of overloading will be introduced to create two or more methods with the same name but different parameters.

The following example explains the same −

Example

Live Demo

```
public class ExampleOverloading {

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      double c = 7.3;
      double d = 9.4;
      int result1 = minFunction(a, b);

      // same function name with different parameters
      double result2 = minFunction(c, d);
      System.out.println("Minimum Value = " + result1);
      System.out.println("Minimum Value = " + result2);
   }

   // for integer
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }

   // for double
   public static double minFunction(double n1, double n2) {
     double min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }
}
```

This will produce the following result −

Output

```
Minimum Value = 6
Minimum Value = 7.3
```

Overloading methods makes program readable. Here, two methods are given by the same name but with different parameters. The minimum number from integer and double types is the result.

Using Command-Line Arguments

Sometimes you will want to pass some information into a program when you run it. This is accomplished by passing command-line arguments to main( ).

A command-line argument is the information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy. They are stored as strings in the String array passed to main( ).

Example

The following program displays all of the command-line arguments that it is called with −

```
public class CommandLine {

   public static void main(String args[]) {
      for(int i = 0; i<args.length; i++) {
         System.out.println("args[" + i + "]: " +  args[i]);
      }
   }
}
```

Try executing this program as shown here −

```
$java CommandLine this is a command line 200 -100
```

This will produce the following result −

Output

```
args[0]: this
args[1]: is
args[2]: a
args[3]: command
args[4]: line
args[5]: 200
args[6]: -100
```

The Constructors

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

Example

Here is a simple example that uses a constructor without parameters −

```
// A simple constructor.
class MyClass {
   int x;

   // Following is the constructor
   MyClass() {
      x = 10;
   }
}
```

You will have to call constructor to initialize objects as follows −

```
public class ConsDemo {

   public static void main(String args[]) {
      MyClass t1 = new MyClass();
      MyClass t2 = new MyClass();
      System.out.println(t1.x + " " + t2.x);
   }
}
```

Output

```
10 10
```

Parameterized Constructor

Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

Example

Here is a simple example that uses a constructor with a parameter −

```
// A simple constructor.
class MyClass {
   int x;

   // Following is the constructor
   MyClass(int i ) {
      x = i;
   }
}
```

You will need to call a constructor to initialize objects as follows −

```
public class ConsDemo {

   public static void main(String args[]) {
      MyClass t1 = new MyClass( 10 );
      MyClass t2 = new MyClass( 20 );
```

```
        System.out.println(t1.x + " " + t2.x);
    }
}
```

This will produce the following result −

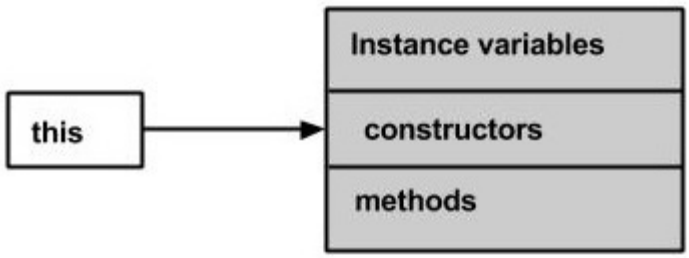
Output

```
10 20
```

The this keyword

this is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor. Using *this* you can refer the members of a class such as constructors, variables and methods.

Note − The keyword *this* is used only within instance methods or constructors



In general, the keyword *this* is used to −

- Differentiate the instance variables from local variables if they have same names, within a constructor or a method.

```
class Student {
    int age;
    Student(int age) {
        this.age = age;
    }
}
```

- Call one type of constructor (parametrized constructor or default) from other in a class. It is known as explicit constructor invocation.

```
class Student {
    int age
    Student() {
        this(20);
    }

    Student(int age) {
        this.age = age;
```

```
    }
}
```

Example

Here is an example that uses *this* keyword to access the members of a class. Copy and paste the following program in a file with the name, **This_Example.java**.

Live Demo

```
public class This_Example {
   // Instance variable num
   int num = 10;

   This_Example() {
      System.out.println("This is an example program on keyword this");
   }

   This_Example(int num) {
      // Invoking the default constructor
      this();

      // Assigning the local variable num to the instance variable num
      this.num = num;
   }

   public void greet() {
      System.out.println("Hi Welcome to Tutorialspoint");
   }

   public void print() {
      // Local variable num
      int num = 20;

      // Printing the local variable
      System.out.println("value of local variable num is : "+num);

      // Printing the instance variable
      System.out.println("value of instance variable num is : "+this.num);

      // Invoking the greet method of a class
      this.greet();
   }

   public static void main(String[] args) {
      // Instantiating the class
      This_Example obj1 = new This_Example();

      // Invoking the print method
      obj1.print();

      // Passing a new value to the num variable through parametrized
constructor
      This_Example obj2 = new This_Example(30);
```

```
      // Invoking the print method again
      obj2.print();
   }
}
```

This will produce the following result −

Output

```
This is an example program on keyword this
value of local variable num is : 20
value of instance variable num is : 10
Hi Welcome to Tutorialspoint
This is an example program on keyword this
value of local variable num is : 20
value of instance variable num is : 30
Hi Welcome to Tutorialspoint
```

Variable Arguments(var-args)

JDK 1.5 enables you to pass a variable number of arguments of the same type to a method. The parameter in the method is declared as follows −

```
typeName... parameterName
```

In the method declaration, you specify the type followed by an ellipsis (...). Only one variable-length parameter may be specified in a method, and this parameter must be the last parameter. Any regular parameters must precede it.

Example

Live Demo

```
public class VarargsDemo {

   public static void main(String args[]) {
      // Call method with variable args
           printMax(34, 3, 3, 2, 56.5);
      printMax(new double[]{1, 2, 3});
   }

   public static void printMax( double... numbers) {
      if (numbers.length == 0) {
         System.out.println("No argument passed");
         return;
      }

      double result = numbers[0];

      for (int i = 1; i <  numbers.length; i++)
      if (numbers[i] >  result)
      result = numbers[i];
      System.out.println("The max value is " + result);
```

```
    }
}
```

This will produce the following result −

Output

```
The max value is 56.5
The max value is 3.0
```

The finalize( ) Method

It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called **finalize( )**, and it can be used to ensure that an object terminates cleanly.

For example, you might use finalize( ) to make sure that an open file owned by that object is closed.

To add a finalizer to a class, you simply define the finalize( ) method. The Java runtime calls that method whenever it is about to recycle an object of that class.

Inside the finalize( ) method, you will specify those actions that must be performed before an object is destroyed.

The finalize( ) method has this general form −

```
protected void finalize( ) {
   // finalization code here
}
```

Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class.

This means that you cannot know when or even if finalize( ) will be executed. For example, if your program ends before garbage collection occurs, finalize( ) will not execute.

Java - OOP Concepts

Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs over Procedure-oriented programming language

1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.

2)OOPs provides data hiding whereas in Procedure-oriented prgramming language a global data can be accessed from anywhere.

3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

In object-oriented programming technique, we design a program using objects and classes.

Object is the physical as well as logical entity whereas class is the logical entity only.

Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengible). The example of integible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

Class in Java

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- **data member**
- **method**

- **constructor**
- **block**
- **class and interface**

Syntax to declare a class:

```
1. class <class_name>{
2.     data member;
3.     method;
4. }
```

Simple Example of Object and Class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

```
1. class Student1{
2.  int id;//data member (also instance variable)
3.  String name;//data member(also instance variable)
4.
5.  public static void main(String args[]){
6.   Student1 s1=new Student1();//creating an object of Student
7.   System.out.println(s1.id);
8.   System.out.println(s1.name);
9.  }
10. }
```

```
Output:0 null
```

Instance variable in Java

A variable that is created inside the class but outside the method, is known as instance variable.Instance variable doesn't get memory at compile time.It gets memory at runtime when object(instance) is created.That is why, it is known as instance variable.

Method in Java

In java, a method is like function i.e. used to expose behaviour of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword

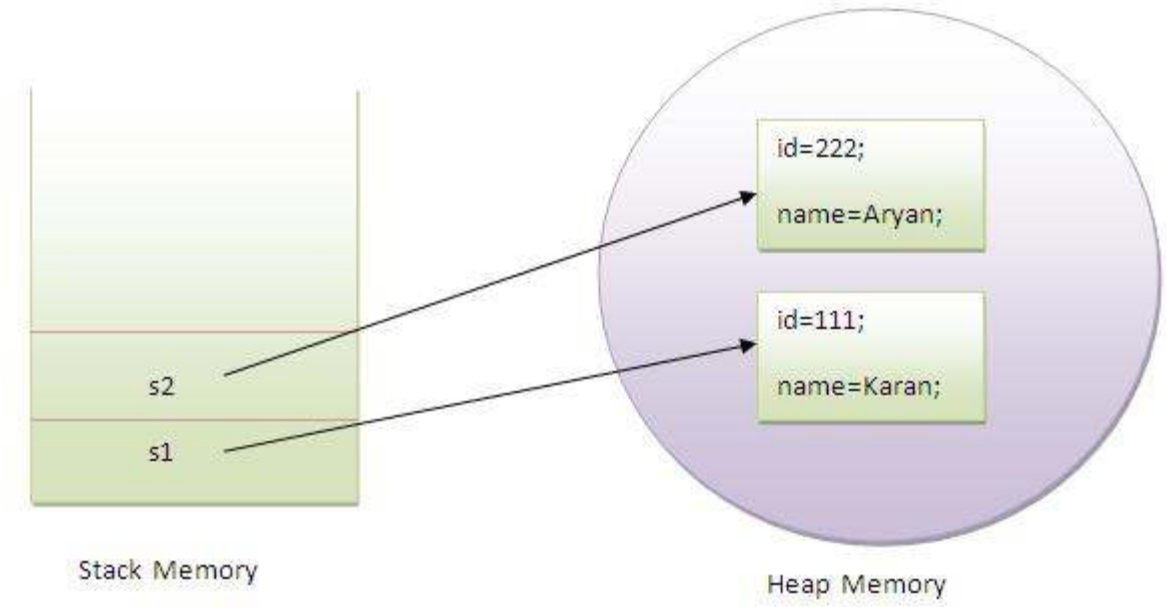
The new keyword is used to allocate memory at runtime.

Example of Object and class that maintains the records of students

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method on it. Here, we are displaying the state (data) of the objects by invoking the displayInformation method.

```
1.  class Student2{
2.   int rollno;
3.   String name;
4.
5.   void insertRecord(int r, String n){  //method
6.    rollno=r;
7.    name=n;
8.   }
9.
10.  void displayInformation(){System.out.println(rollno+" "+name);}//method
11.
12.  public static void main(String args[]){
13.   Student2 s1=new Student2();
14.   Student2 s2=new Student2();
15.
16.   s1.insertRecord(111,"Karan");
17.   s2.insertRecord(222,"Aryan");
18.
19.   s1.displayInformation();
20.   s2.displayInformation();
21.
22.  }
23. }
```

```
111 Karan
222 Aryan
```



As you see in the above figure, object gets the memory in Heap area and reference variable refers to the object allocated in the Heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

Another Example of Object and Class

There is given another example that maintains the records of Rectangle class. Its exaplanation is same as in the above Student class example.

```
1.  class Rectangle{
2.   int length;
3.   int width;
4.
5.   void insert(int l,int w){
6.    length=l;
7.    width=w;
8.   }
9.
10.  void calculateArea(){System.out.println(length*width);}
11.
12.  public static void main(String args[]){
```

```
13.   Rectangle r1=new Rectangle();
14.   Rectangle r2=new Rectangle();
15.
16.   r1.insert(11,5);
17.   r2.insert(3,15);
18.
19.   r1.calculateArea();
20.   r2.calculateArea();
21. }
22. }
```

```
Output:55
       45
```

Annonymous object

Annonymous simply means nameless.An object that have no reference is known as annonymous object.

If you have to use an object only once, annonymous object is a good approach.

```
1.   class Calculation{
2.
3.    void fact(int  n){
4.     int fact=1;
5.     for(int i=1;i<=n;i++){
6.      fact=fact*i;
7.     }
8.    System.out.println("factorial is "+fact);
9.   }
10.
11.  public static void main(String args[]){
12.   new Calculation().fact(5);//calling method with annonymous object
13. }
14. }
```

```
Output:Factorial is 120
```

Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

```
1.   Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
```

Let's see the example:

```
1.  class Rectangle{
2.   int length;
3.   int width;
4.
5.   void insert(int l,int w){
6.    length=l;
7.    width=w;
8.   }
9.
10.  void calculateArea(){System.out.println(length*width);}
11.
12.  public static void main(String args[]){
13.   Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
14.
15.   r1.insert(11,5);
16.   r2.insert(3,15);
17.
18.   r1.calculateArea();
19.   r2.calculateArea();
20. }
21. }
```

```
Output:55
       45
```

Java static keyword

The **static keyword** in java is used for memory management mainly. We can apply java 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:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

---

1) Java 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

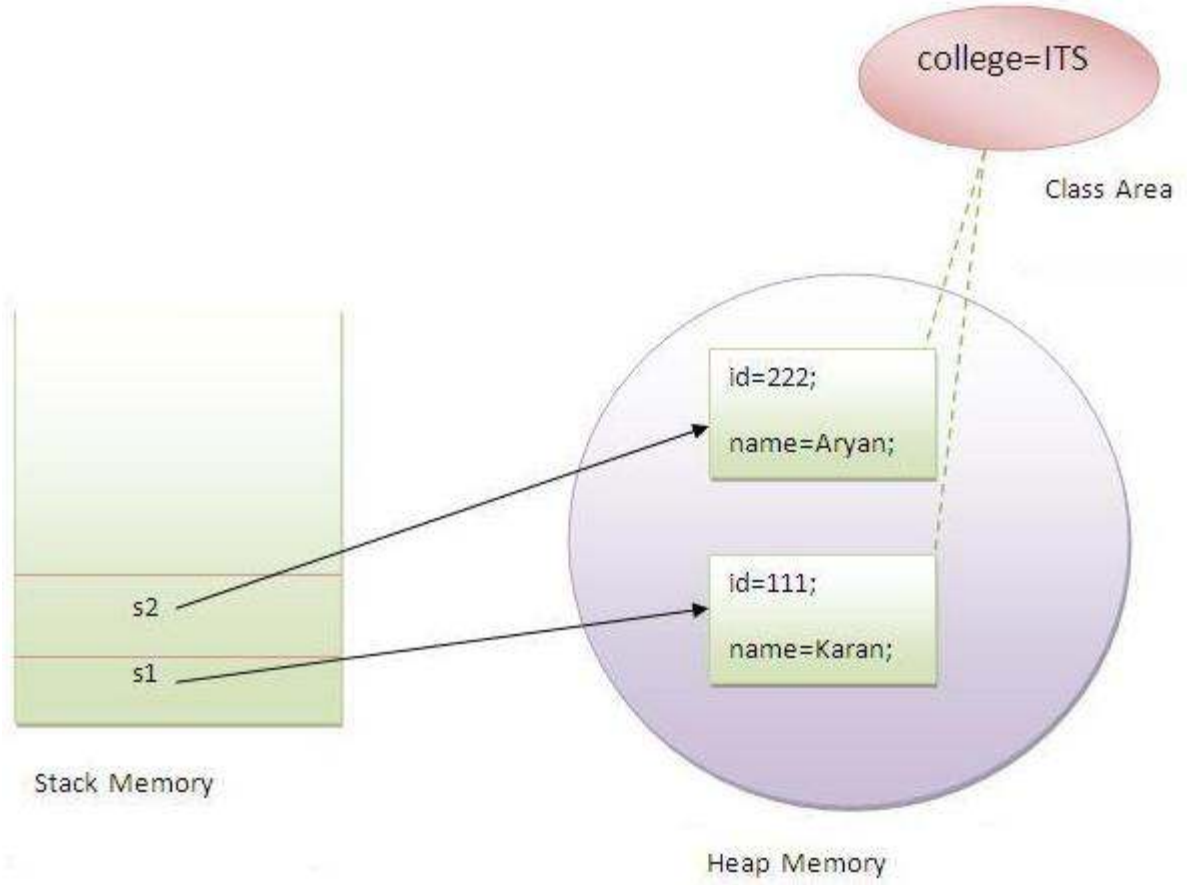
It makes your program **memory efficient** (i.e it saves memory).

Java static property is shared to all objects.

Example of static variable

```
1.  //Program of static variable
2.
3.  class Student8{
4.     int rollno;
5.     String name;
6.     static String college ="ITS";
7.
8.     Student8(int r,String n){
9.     rollno = r;
10.    name = n;
11.    }
12.  void display (){System.out.println(rollno+" "+name+" "+college);}
13.
14.  public static void main(String args[]){
15.  Student8 s1 = new Student8(111,"Karan");
16.  Student8 s2 = new Student8(222,"Aryan");
17.
18.  s1.display();
19.  s2.display();
20.  }
21. }
```

```
Output:111 Karan ITS
       222 Aryan ITS
```



2) Java 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

```
1.  //Program of changing the common property of all objects(static field).
2.
3.  class Student9{
4.      int rollno;
5.      String name;
6.      static String college = "ITS";
7.
8.      static void change(){
9.      college = "BBDIT";
10.     }
11.
12.     Student9(int r, String n){
```

```
13.     rollno = r;
14.     name = n;
15.     }
16.
17.     void display (){System.out.println(rollno+" "+name+" "+college);}
18.
19.     public static void main(String args[]){
20.     Student9.change();
21.
22.     Student9 s1 = new Student9 (111,"Karan");
23.     Student9 s2 = new Student9 (222,"Aryan");
24.     Student9 s3 = new Student9 (333,"Sonoo");
25.
26.     s1.display();
27.     s2.display();
28.     s3.display();
29.     }
30. }
```

```
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:

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

```
1.  class A{
2.   int a=40;//non static
3.
4.   public static void main(String args[]){
5.    System.out.println(a);
6.   }
7.  }
```

```
Output:Compile Time Error
```

---

3) Java 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

```
1. class A2{
2.   static{System.out.println("static block is invoked");}
3.   public static void main(String args[]){
4.    System.out.println("Hello main");
5.   }
6. }
```

```
Output:static block is invoked
       Hello main
```

---

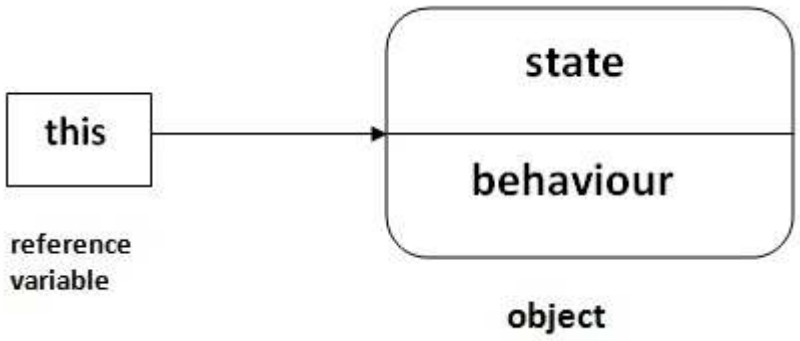
this keyword in java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.



---

1) The this keyword can be used to refer current class instance variable.

If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of

ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
1.  class Student10{
2.      int id;
3.      String name;
4.
5.      Student10(int id,String name){
6.      id = id;
7.      name = name;
8.      }
9.      void display(){System.out.println(id+" "+name);}
10.
11.     public static void main(String args[]){
12.     Student10 s1 = new Student10(111,"Karan");
13.     Student10 s2 = new Student10(321,"Aryan");
14.     s1.display();
15.     s2.display();
16.     }
17. }
```

```
Output:0 null
       0 null
```

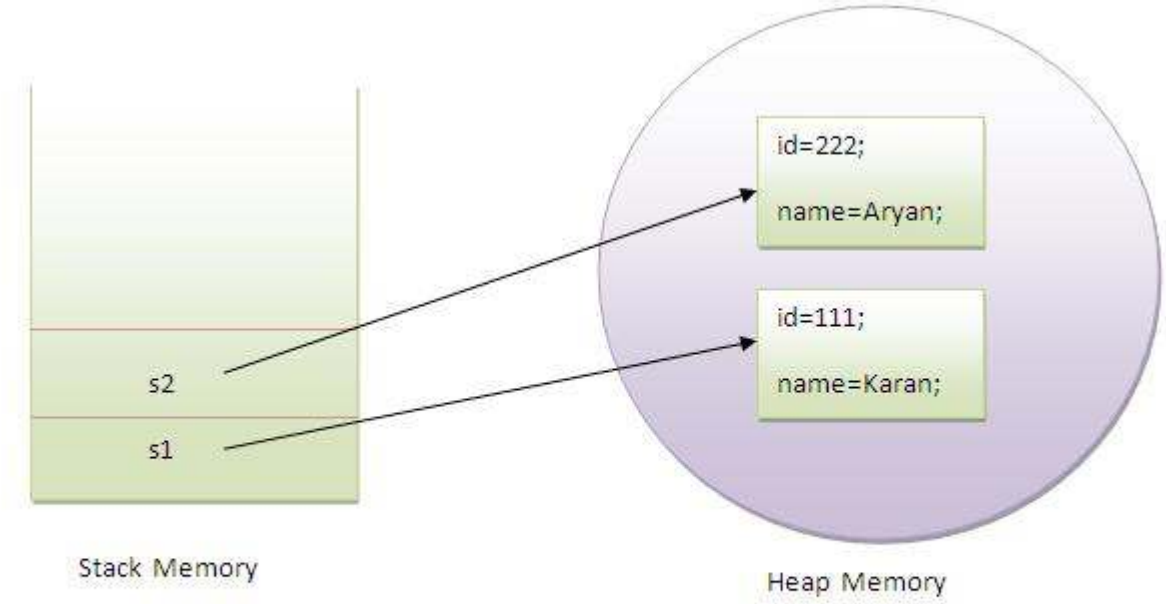
In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable.

Solution of the above problem by this keyword

```
1.  //example of this keyword
2.  class Student11{
3.      int id;
4.      String name;
5.
6.      Student11(int id,String name){
7.      this.id = id;
8.      this.name = name;
9.      }
10.     void display(){System.out.println(id+" "+name);}
11.     public static void main(String args[]){
12.     Student11 s1 = new Student11(111,"Karan");
13.     Student11 s2 = new Student11(222,"Aryan");
14.     s1.display();
15.     s2.display();
16. }
```

```
17. }
```

```
Output111 Karan
       222 Aryan
```



If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

```
1.  class Student12{
2.      int id;
3.      String name;
4.
5.      Student12(int i,String n){
6.      id = i;
7.      name = n;
8.      }
9.      void display(){System.out.println(id+" "+name);}
10.     public static void main(String args[]){
11.     Student12 e1 = new Student12(111,"karan");
12.     Student12 e2 = new Student12(222,"Aryan");
13.     e1.display();
14.     e2.display();
15. }
16. }
```

Test it Now

```
Output:111 Karan
       222 Aryan
```

---

2) this() can be used to invoked current class constructor.

The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

```
1.  //Program of this() constructor call (constructor chaining)
2.
3.  class Student13{
4.     int id;
5.     String name;
6.     Student13(){System.out.println("default constructor is invoked");}
7.
8.     Student13(int id,String name){
9.     this ();//it is used to invoked current class constructor.
10.    this.id = id;
11.    this.name = name;
12.    }
13.    void display(){System.out.println(id+" "+name);}
14.
15.    public static void main(String args[]){
16.    Student13 e1 = new Student13(111,"karan");
17.    Student13 e2 = new Student13(222,"Aryan");
18.    e1.display();
19.    e2.display();
20.   }
21. }
```

```
Output:
default constructor is invoked
default constructor is invoked
       111 Karan
       222 Aryan
```

Where to use this() constructor call?

The this() constructor call should be used to reuse the constructor in the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
1.  class Student14{
2.     int id;
```

```
3.      String name;
4.      String city;
5.
6.      Student14(int id,String name){
7.      this.id = id;
8.      this.name = name;
9.      }
10.     Student14(int id,String name,String city){
11.     this(id,name);//now no need to initialize id and name
12.     this.city=city;
13.     }
14.     void display(){System.out.println(id+" "+name+" "+city);}
15.
16.     public static void main(String args[]){
17.     Student14 e1 = new Student14(111,"karan");
18.     Student14 e2 = new Student14(222,"Aryan","delhi");
19.     e1.display();
20.     e2.display();
21.    }
22. }
```

```
Output:111 Karan null
       222 Aryan delhi
```

Rule: Call to this() must be the first statement in constructor.

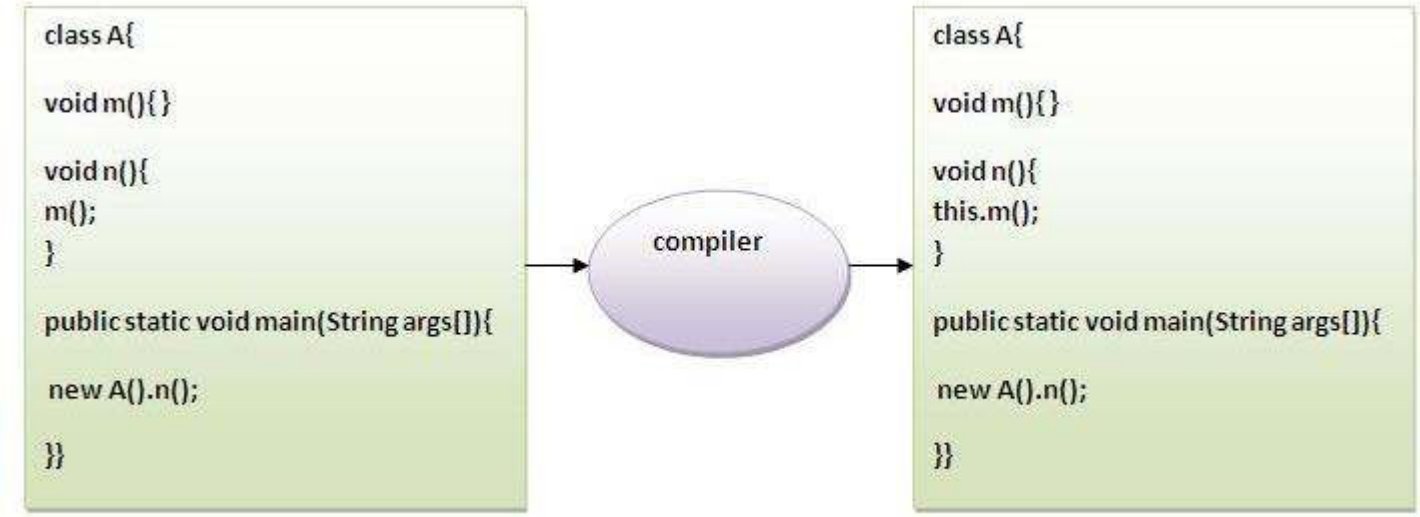
```
1.   class Student15{
2.      int id;
3.      String name;
4.      Student15(){System.out.println("default constructor is invoked");}
5.
6.      Student15(int id,String name){
7.      id = id;
8.      name = name;
9.      this ();//must be the first statement
10.     }
11.     void display(){System.out.println(id+" "+name);}
12.
13.     public static void main(String args[]){
14.     Student15 e1 = new Student15(111,"karan");
15.     Student15 e2 = new Student15(222,"Aryan");
16.     e1.display();
17.     e2.display();
18.    }
19. }
```

```
Output:Compile Time Error
```

---

3)The this keyword can be used to invoke current class method (implicitly).

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example




```
1.  class S{
2.    void m(){
3.    System.out.println("method is invoked");
4.    }
5.    void n(){
6.    this.m();//no need because compiler does it for you.
7.    }
8.    void p(){
9.    n();//complier will add this to invoke n() method as this.n()
10.   }
11.   public static void main(String args[]){
12.   S s1 = new S();
13.   s1.p();
14.   }
15. }
```

```
Output:method is invoked
```

---

4) this keyword can be passed as an argument in the method.

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
1.  class S2{
2.    void m(S2 obj){
3.    System.out.println("method is invoked");
4.    }
5.    void p(){
6.    m(this);
7.    }
8.
9.    public static void main(String args[]){
10.   S2 s1 = new S2();
11.   s1.p();
12.   }
13. }
```

```
Output:method is invoked
```

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one.

---

5) The this keyword can be passed as argument in the constructor call.

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
1.  class B{
2.    A4 obj;
3.    B(A4 obj){
4.      this.obj=obj;
5.    }
6.    void display(){
7.      System.out.println(obj.data);//using data member of A4 class
8.    }
9.  }
10.
11. class A4{
12.   int data=10;
13.   A4(){
14.    B b=new B(this);
15.    b.display();
16.   }
17.   public static void main(String args[]){
18.    A4 a=new A4();
19.   }
```

```
20. }
```

```
Output:10
```

---

6) The this keyword can be used to return current class instance.

We can return the this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

```
1. return_type method_name(){
2. return this;
3. }
```

Example of this keyword that you return as a statement from the method

```
1. class A{
2. A getA(){
3. return this;
4. }
5. void msg(){System.out.println("Hello java");}
6. }
7.
8. class Test1{
9. public static void main(String args[]){
10. new A().getA().msg();
11. }
12. }
```

```
Output:Hello java
```

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
1. class A5{
2. void m(){
3. System.out.println(this);//prints same reference ID
4. }
5.
6. public static void main(String args[]){
```

```
7.  A5 obj=new A5();
8.  System.out.println(obj);//prints the reference ID
9.
10. obj.m();
11. }
12. }
```

```
Output:A5@22b3ea59
       A5@22b3ea59
```

super keyword in java

The **super** keyword in java is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of java super Keyword

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method.

1) super is used to refer immediate parent class instance variable.

Problem without super keyword

```
1.  class Vehicle{
2.    int speed=50;
3.  }
4.  class Bike3 extends Vehicle{
5.    int speed=100;
6.    void display(){
7.     System.out.println(speed);//will print speed of Bike
8.    }
9.    public static void main(String args[]){
10.   Bike3 b=new Bike3();
11.   b.display();
12. }
13. }
```

```
Output:100
```

In the above example Vehicle and Bike both class have a common property speed. Instance variable of current class is refered by instance bydefault, but I have to refer parent class instance variable that is why we use super keyword to distinguish between parent class instance variable and current class

instance variable.

Solution by super keyword

```
1.  //example of super keyword
2.
3.  class Vehicle{
4.    int speed=50;
5.  }
6.
7.  class Bike4 extends Vehicle{
8.    int speed=100;
9.
10.   void display(){
11.    System.out.println(super.speed);//will print speed of Vehicle now
12.   }
13.   public static void main(String args[]){
14.    Bike4 b=new Bike4();
15.    b.display();
16.
17. }
18. }
```

```
Output:50
```

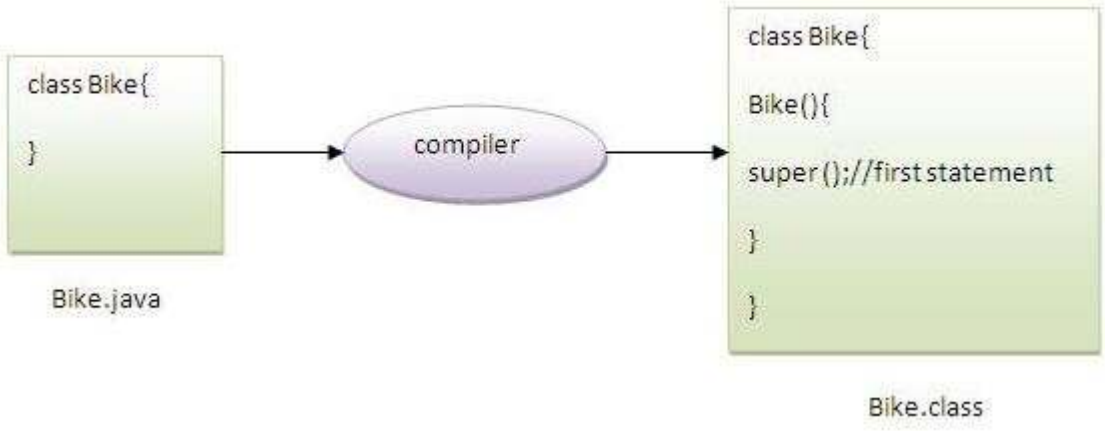
2) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor as given below:

```
1.  class Vehicle{
2.    Vehicle(){System.out.println("Vehicle is created");}
3.  }
4.
5.  class Bike5 extends Vehicle{
6.    Bike5(){
7.     super();//will invoke parent class constructor
8.     System.out.println("Bike is created");
9.    }
10.   public static void main(String args[]){
11.    Bike5 b=new Bike5();
12.
13. }
14. }
```

```
Output:Vehicle is created
       Bike is created
```

Note: super() is added in each class constructor automatically by compiler.



As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement.If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

Another example of super keyword where super() is provided by the compiler implicitly.

```
1.  class Vehicle{
2.    Vehicle(){System.out.println("Vehicle is created");}
3.  }
4.
5.  class Bike6 extends Vehicle{
6.    int speed;
7.    Bike6(int speed){
8.      this.speed=speed;
9.      System.out.println(speed);
10.   }
11.   public static void main(String args[]){
12.    Bike6 b=new Bike6(10);
13.  }
14. }
```

```
Output:Vehicle is created
       10
```

3) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
1.  class Person{
2.  void message(){System.out.println("welcome");}
```

```
}

class Student16 extends Person{
void message(){System.out.println("welcome to java");}

void display(){
message();//will invoke current class message() method
super.message();//will invoke parent class message() method
}

public static void main(String args[]){
Student16 s=new Student16();
s.display();
}
}
```

```
Output:welcome to java
welcome
```

In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.

Program in case super is not required

```
class Person{
void message(){System.out.println("welcome");}
}

class Student17 extends Person{

void display(){
message();//will invoke parent class message() method
}

public static void main(String args[]){
Student17 s=new Student17();
s.display();
}
}
```

```
Output:welcome
```