**Object Oriented Features**

* Class
* Object
* Inheritance
* Polymorphism
* Encapsulation
* Abstraction

**Programming languages based on object oriented features**

**Object Oriented Programming Languages**

* Object oriented programming languages supports all the object oriented features.

Example: Java

**Object Based Programming languages**

* Object based programming languages supports all most all the object oriented features excluding "Inheritance".

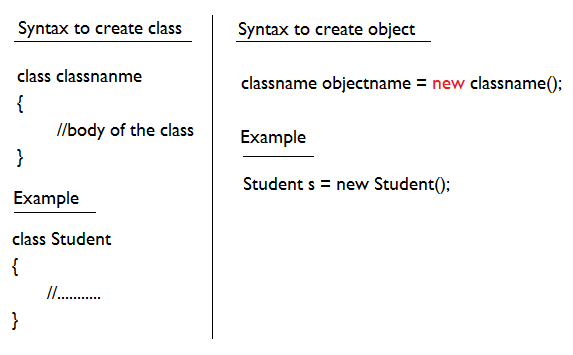
Example: Java Script

**Class**

* Class is a user defined data type.
* Class is a plan or blue print.

**Object**

* Object is instance of the class.



**Example#1**

class Student{

int age;

String name;

String address;

}

class Test{

public static void main(String args[]){

Student s = new Student();

System.out.println("Age = "+s.age);

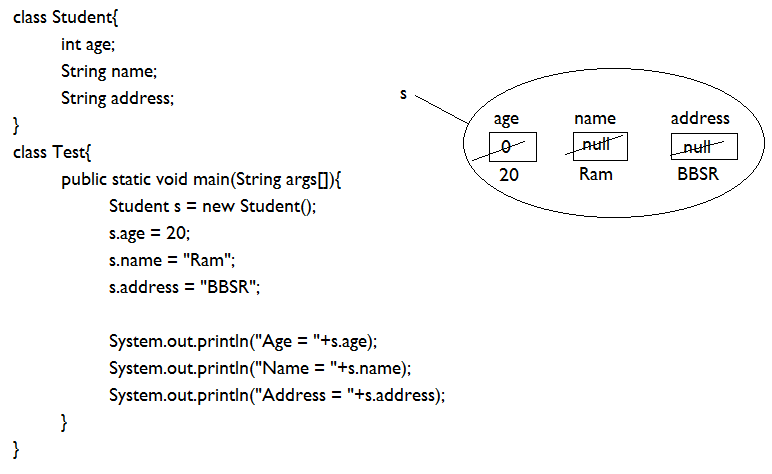
System.out.println("Name = "+s.name);

System.out.println("Address = "+s.address);

}

}

**Example#2**

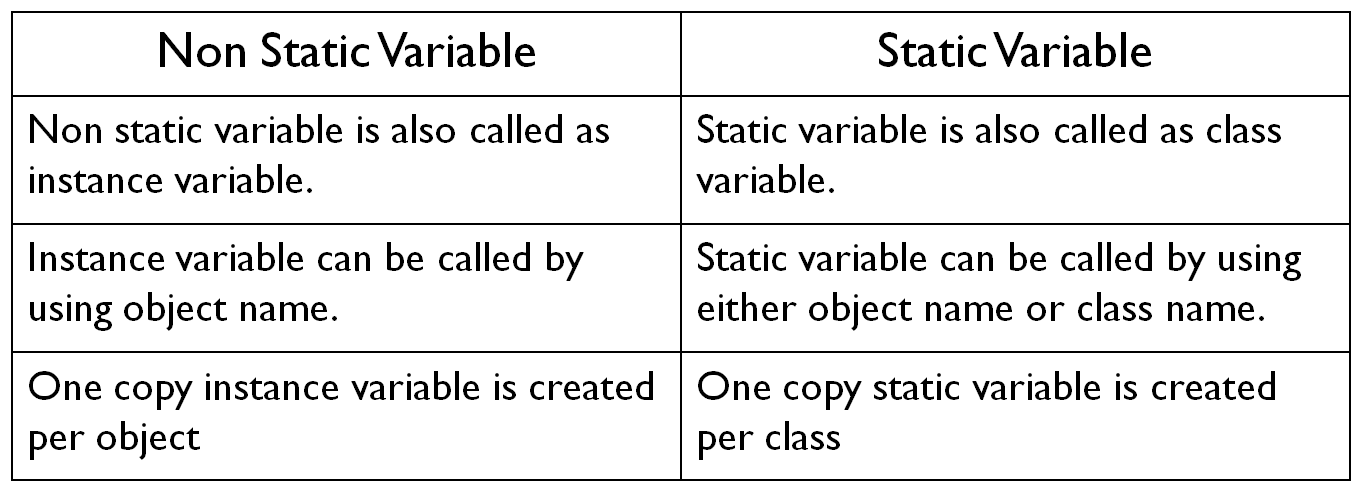


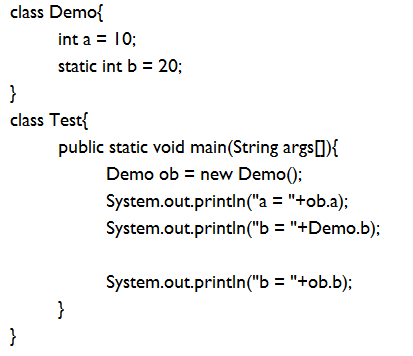
**Class Elements**

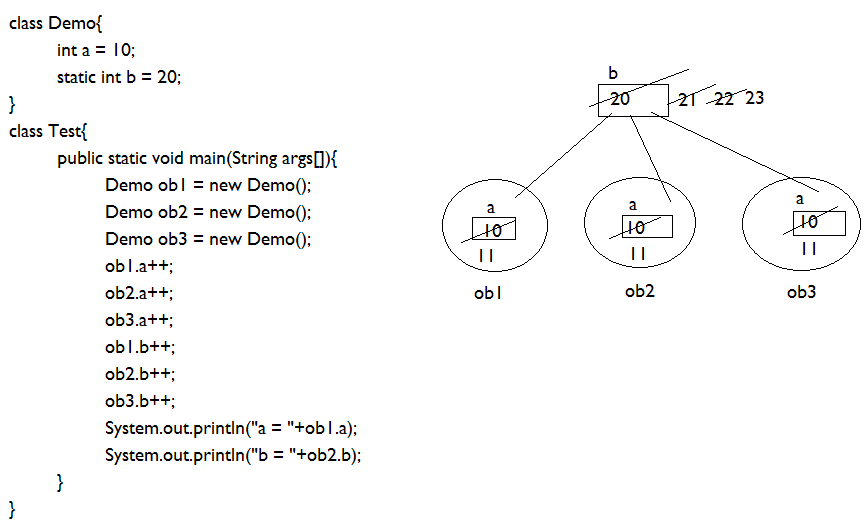
1. variables
2. methods
3. constructors
4. instance blocks
5. static blocks

**Variable**

**Non Static Variable vs. Static Variable**







**Methods**

* Methods are used to write the business logics of the project.
* We can declare any number of methods inside the class based on the requirement.

**Types of Method**

* Instance method/ non static method
* Static method

**Instance method**

class Test

{

void show()

{

System.out.println("Inside instance method...");

}

}

class Main

{

public static void main(String args[])

{

Test ob = new Test();

ob.show();

}

}

**Static Method**

class Test

{

static void show()

{

System.out.println("Inside static method...");

}

}

class Main{

public static void main(String args[])

{

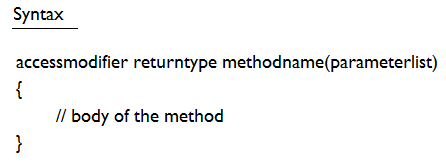
Test.show();

}

}

**Method Signature**

* Method signature includes the name of the method and parameters list. Return type and modifiers list is not part of a method signature.
* Every method contains two parts.
  + Method declaration
  + Method implementation



**Different ways to create method**

* No Return No Argument
* No Return with argument
* With Return No Argument
* With return With Argument

**No Return No Argument**

class Test

{

void show()

{

System.out.println("No retutn, No argument");

}

}

class Main

{

public static void main(String args[])

{

Test ob = new Test();

ob.show();

}

}

**No Return with Argument**

class Test

{

void add(int x, int y)

{

System.out.println("Sum = "+(x+y));

}

}

class Main

{

public static void main(String args[])

{

Test ob = new Test();

ob.add(10,20);

}

}

**With Return No Argument**

class Test

{

int currentYear()

{

return 2022;

}

}

class Main

{

public static void main(String args[])

{

Test ob = new Test();

int x = ob.currentYear();

System.out.println(x);

}

}

**With Return With Argument**

class Test{

int add(int x, int y){

return x+y;

}

}

class Main{

public static void main(String args[]){

Test ob = new Test();

int res = ob.add(10,20);

System.out.println("Sum = "+res);

}

}

**Method Overloading**

* A class having more than one method with same name and different argument lists is called method overloading.
* The argument lists of the methods must differ in either of these ways:
  + Number of parameters
  + Type of parameters
  + Order of parameters

**Example#1 (number of parameters)**

class Test{

void add(int x, int y){

System.out.println(x+y);

}

void add(int x, int y, int z){

System.out.println(x+y+z);

}

}

class MethodOverloading{

public static void main(String args[]){

Test ob = new Test();

ob.add(10,20);

ob.add(10,20,30);

}

}

**Example#2 (type of parameters)**

class Test{

void add(int x, int y){

System.out.println(x+y);

}

void add(float x, float y){

System.out.println(x+y);

}

}

class MethodOverloading{

public static void main(String args[]){

Test ob = new Test();

ob.add(10,20);

ob.add(10.2f, 20.2f);

}

}

**Example#3 (Order of Parameter)**

class Test{

void add(int x, float y){

System.out.println(x+y);

}

void add(float x, int y){

System.out.println(x+y);

}

}

class MethodOverloading{

public static void main(String args[]){

Test ob = new Test();

ob.add(10,20.2f);

ob.add(10.2f, 30);

}

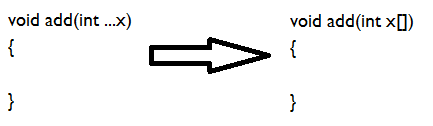
}

**Note:**

* Method overloading is called compile time polymorphism.
* Poly – Many
* Morph – Form
* Polymorphism means one interface (method) with multiple forms.

**VarArg Method**

* Method which can take any number of arguments is called VarArg method.



**Example#1**

class Test{

void add(int ...x){

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

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

}

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

}

}

class Main{

public static void main(String args[]){

Test t = new Test();

t.add();

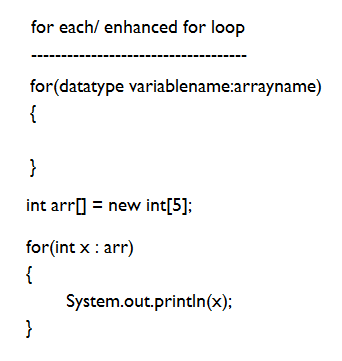
t.add(10);

t.add(10,20);

t.add(10,20,30);

}

}



**Example#2**

class Test{

void add(int ...x){

int sum = 0;

for(int a:x){

sum = sum + a;

}

System.out.println("Sum = "+sum);

}

}

class Main{

public static void main(String args[]){

Test t = new Test();

t.add();

t.add(10);

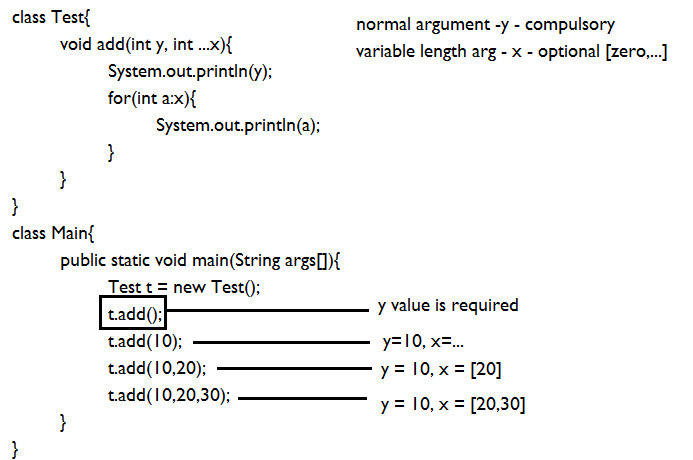
t.add(10,20);

t.add(10,20,30);

}

}

**Example#3**



**Constructor**

* A constructor is a special method whose name must be same as the class name.
* Constructor must not have return type.
* Constructor is used to initialize objects.
* The constructor is called when an object of a class is created.

**Example**

class Test{

Test(){

System.out.println("constructor...");

}

void show(){

System.out.println("method...");

}

}

class ConstructorDemo{

public static void main(String args[]){

Test t = new Test();

}

}

**Types of Constructor**

* 0-arg Constructor
* Parameterized Constructor

**0-arg Constructor**

* A constructor that is created without parameters is called 0-arg constructor.

**Example#1**

class Test{

Test(){

System.out.println("constructor...");

}

}

class ConstructorDemo{

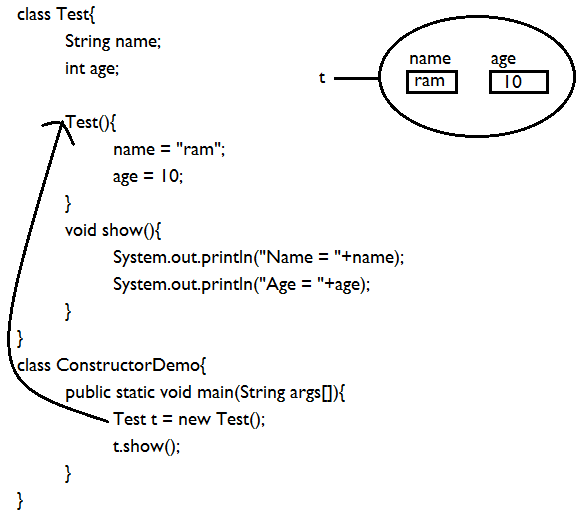
public static void main(String args[]){

Test t = new Test();

}

}

**Example#2**



**Example#3**

class Student{

String name;

int age;

Student(){

name = "ram";

age = 10;

}

void show(){

System.out.println("Name = "+name);

System.out.println("Age = "+age);

}

}

class ConstructorDemo{

public static void main(String args[]){

Student s1 = new Student();

Student s2 = new Student();

Student s3 = new Student();

s1.show();

s2.show();

s3.show();

}

}

**Parameterized Constructor**

* A constructor that is created with one or more parameters is called parameterized constructor.

**Example**

class Student{

String name;

int age;

Student(String name, int age){

this.name = name;

this.age = age;

}

void show(){

System.out.println("Name = "+name);

System.out.println("Age = "+age);

}

}

class ConstructorDemo{

public static void main(String args[]){

Student s1 = new Student("Ram",10);

Student s2 = new Student("Raj",20);

Student s3 = new Student("Raja",30);

s1.show();

s2.show();

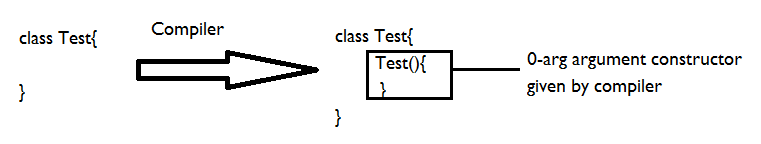
s3.show();

}

}

**Default Constructor**

* If we are not writing constructor for a class then compiler generates one constructor(0-argument) that constructor is called default constructor.



**Example**

class Test{

}

class Main{

public static void main(String args[]){

Test t = new Test();

}

}

**Note:**

* Compiler will not provide default constructor in presence of parameterized constructor.

**Example**

class Test{

Test(){

}

Test(int x){

System.out.println(x);

}

}

class Main{

public static void main(String args[]){

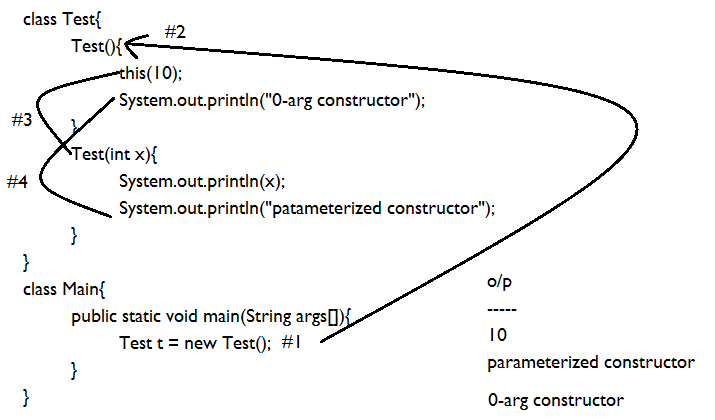
Test t = new Test();

}

}

**Constructor chaining**

* One constructor is calling same class constructor is called constructor calling.
* We are achieving constructor calling by using *this* keyword.
* Inside constructor we are able to declare only one this keyword that must be first statement of the constructor.



**Constructor Overloading**

* Constructor Overloading is a feature that allows a class to have more than one constructor having the same name and different argument lists.
* Argument list can be different based on:
  + Number of parameter
  + Type of parameter
  + Order of parameter

**Example#1**

class Test{

Test(int x,int y){

System.out.println(x+y);

}

Test(int x,int y,int z){

System.out.println(x+y+z);

}

}

class Main{

public static void main(String args[]){

Test t = new Test(10,20);

Test t = new Test(10,20,30);

}

}

**Example#2**

class Test{

Test(int x,int y){

System.out.println(x+y);

}

Test(float x,float y){

System.out.println(x+y);

}

}

class Main{

public static void main(String args[]){

Test t = new Test(10,20);

Test t = new Test(10.2f,20.3f);

}

}

**Example#3**

class Test{

Test(int x,float y){

System.out.println(x+y);

}

Test(float x,int y){

System.out.println(x+y);

}

}

class Main{

public static void main(String args[]){

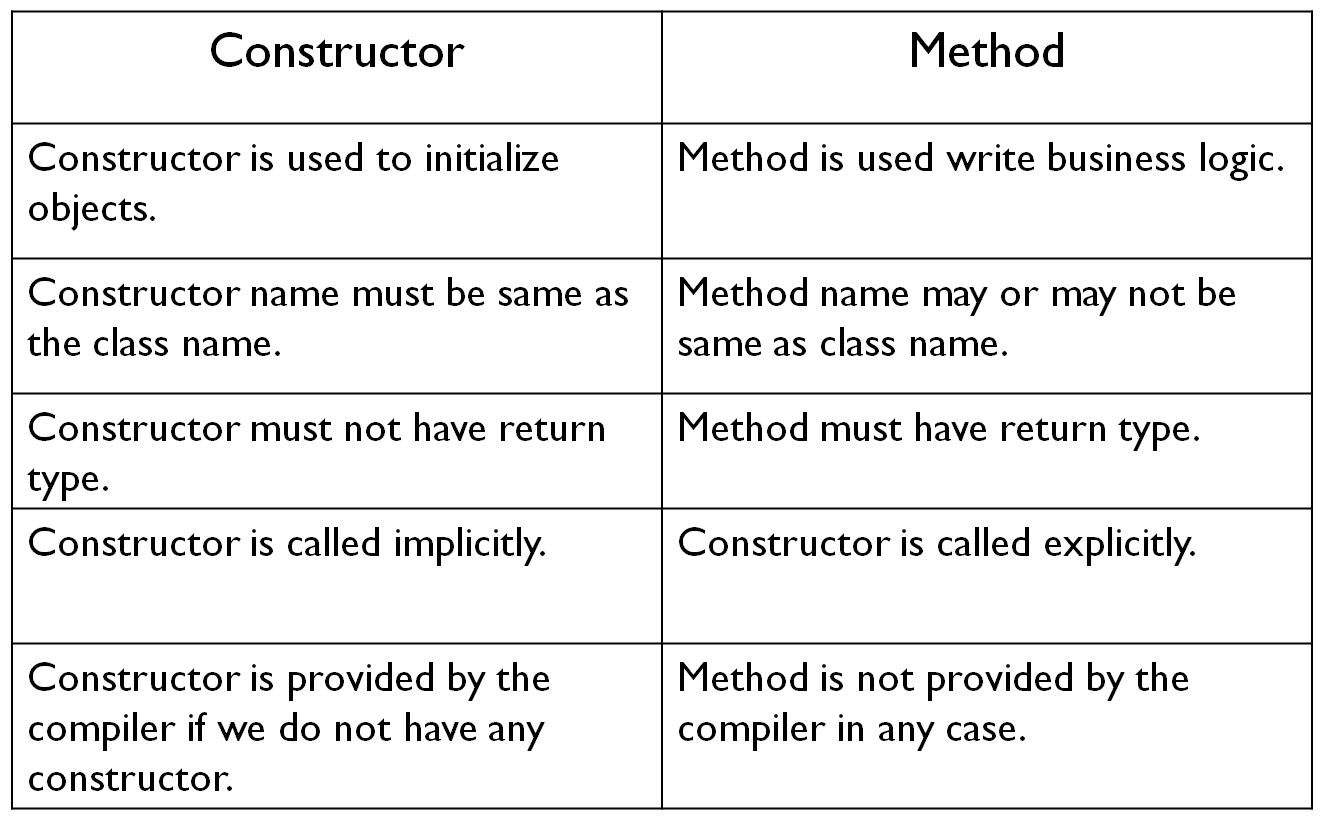
Test t = new Test(10,20.2f);

Test t = new Test(10.2f,20);

}

}

**Constructor vs. Method**



**Instance Blocks**

* Instance blocks are executed during object creation just before constructor execution.
* Instance blocks execution depends on object creation it means if we are creating 10 objects 10 times instance blocks are executed.

**Example#2**

class Test{

{

System.out.println("instance block");

}

Test(){

System.out.println("constructor...");

}

Test(int x){

System.out.println("parameterized constructor");

}

}

class Main{

public static void main(String args[]){

Test t1 = new Test();

Test t2 = new Test();

Test t3 = new Test(10);

}

}

**Note:**

* If we want to provide common logic to all constructors of a class then we will go for instance block.

**Example**

class Test{

static int count = 0;

{

count++;

}

Test(){

}

Test(int x){

}

}

class Main{

public static void main(String args[]){

Test t1 = new Test();

Test t2 = new Test();

Test t3 = new Test(10);

System.out.println("No. of object = "+Test.count);

}

}

**Static block**

* Static blocks are used to initialize the static data member.
* Static block is executed before the main method at the time of class loading.

**Example**

class Main{

static int x;

static

{

x = 10;

System.out.println("static block");

}

public static void main(String args[]){

System.out.println("main method");

}

}