**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".

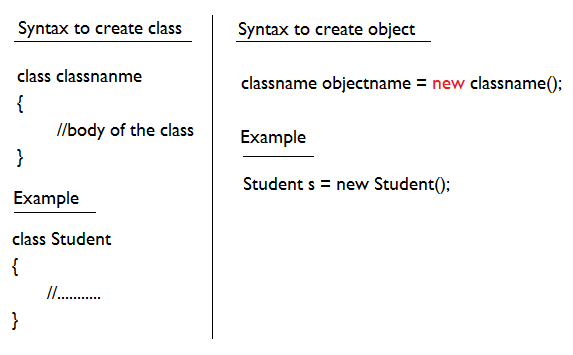
Example: Java Script

**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);

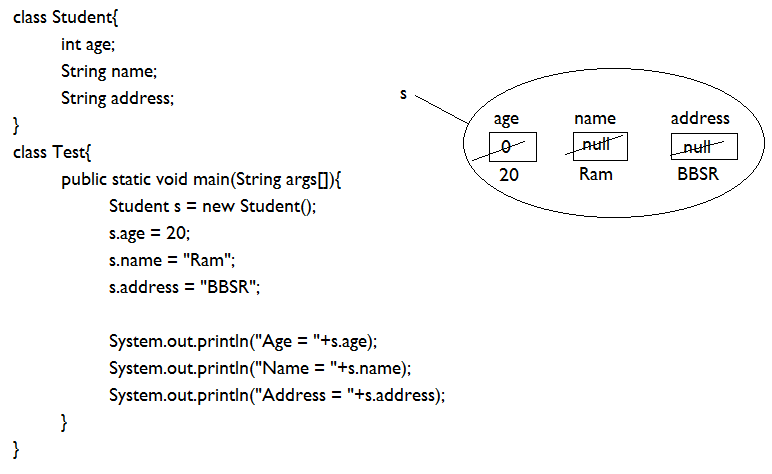
System.out.println("Name = "+s.name);

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

}

}

**Example#2**

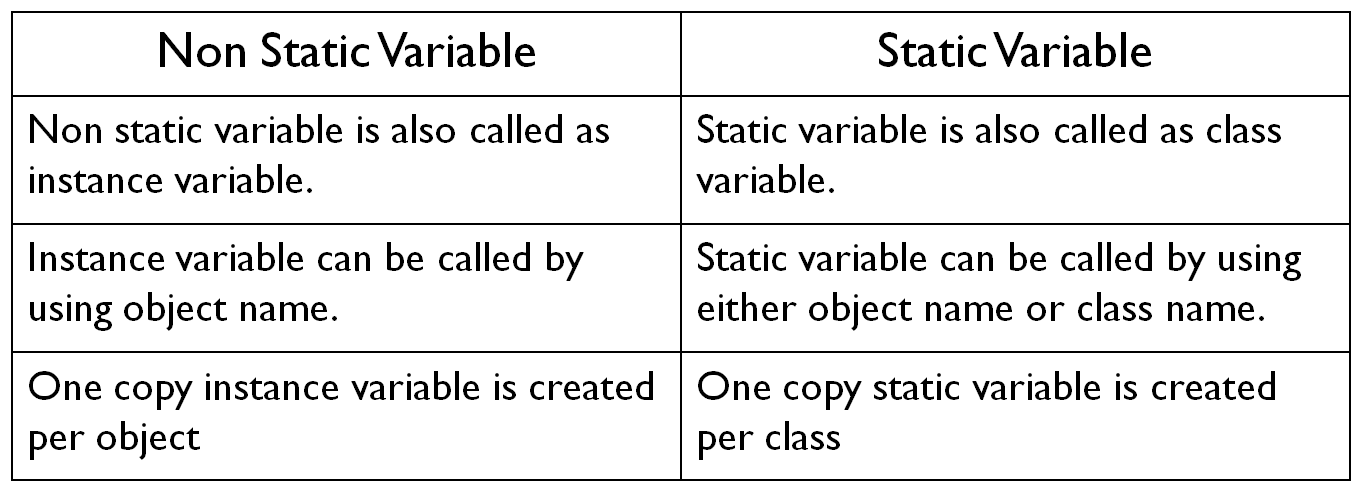


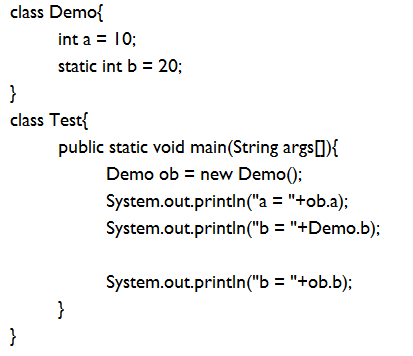
**Class Elements**

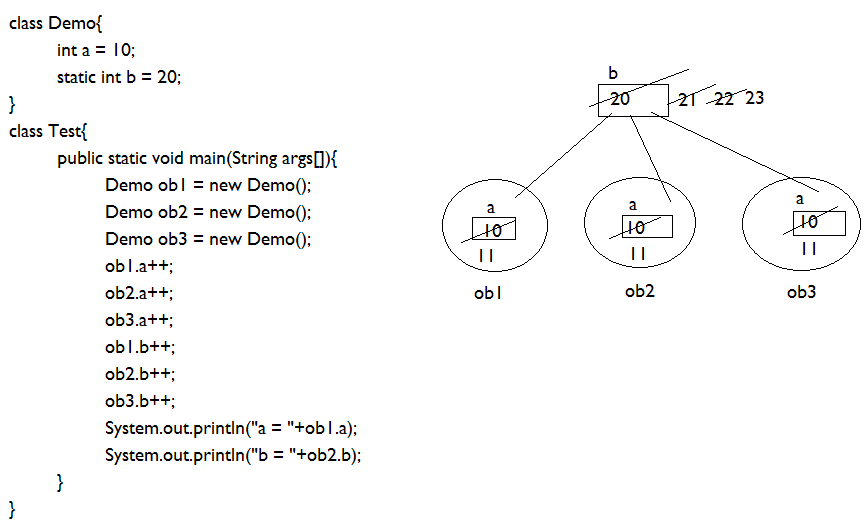
1. variables
2. methods
3. constructors
4. instance blocks
5. static blocks

**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[])

{

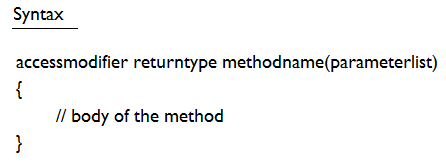
Test.show();

}

}

**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.