Getter : It should not have any parameters.

Ex: int getAge(){

}

Setter : It can have parameters.

Ex : int setAge(int age)

{

this.age=age;

}

For Boolean getter:

Ex:public boolean isMarried()

{

}

A computer screen with text and images

Description automatically generated

This will hold the reference of the Currently executing object.

Let suppose there are multiple Objects as shown in the above example.

this will hold the reference of the currently executing object.

At point it will hold the reference of the st1 object and other point it will hold st2 Object.

For One class we can create many objects.

A computer screen shot of a program

Description automatically generated

A computer screen shot of a blue screen

Description automatically generated

We can have common setter for all the data members.

Constructor :

Constructor name will be similar to method name.

Constructor will be called when the Object is created.

Constructor doesn’t have any return type.

Student st1=new Student();

When there are some statements that has be executed during Object Creation we can keep them in Constructor.

Constructor is called at the time of Object Creation/instantiation.

If the user doesn't specify constructor in the class by default JVM will provide a default constructor.

If class already has some constructor default constructor will not be provided.

**class** StudentNew

{

**private** **int** age;

**private** String name; ///Data Members//

**private** String city; //If data members are made as private then that variables cannot be accessed out

//side the class in which it is pre

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

}

**public** **class** LaunchStudent {

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

StudentNew s=**new** StudentNew();

s.setAge(10);

System.***out***.println(s.getAge()); }}

In the above example if we see there is no constructor then JVM by will include default constructor.

Default constructor ---Constructor with Zero Parameters.

If we include any constructor, then JVM will not include any constructor. Then

Constructor Overloading : Multiple Constructors having same name but difference in the number/type of parameters.

**class** StudentNew

{

**private** **int** age;

**private** String name; ///Data Members//

**private** String city; //If data members are made as private then that variables cannot be accessed out

//side the class in which it is pre

**public** **int** getAge() {

**return** age;

}

**public** StudentNew(**int** age, String name, String city) {

**this**.age = age;

**this**.name = name;

**this**.city = city;

}

**public** StudentNew()

{

age=20;

name="Chinmay";

city="Madanapalle";

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

}

**public** **class** ConstructoOverload {

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

StudentNew s=**new** StudentNew();

System.***out***.println(s.getAge());

StudentNew s1=**new** StudentNew();

}

}

Object Class is the parent class for all the Class.

In an Constructor the first line will be call lo parent class constructor using super() even if we don’t specify

. **public** StudentNew()

{

age=20;

name="Chinmay";

city="Madanapalle";

}

If we don’t specify anything then super() will inserted in the first line of constructor.

**public** StudentNew()

{

**super**();

age=20;

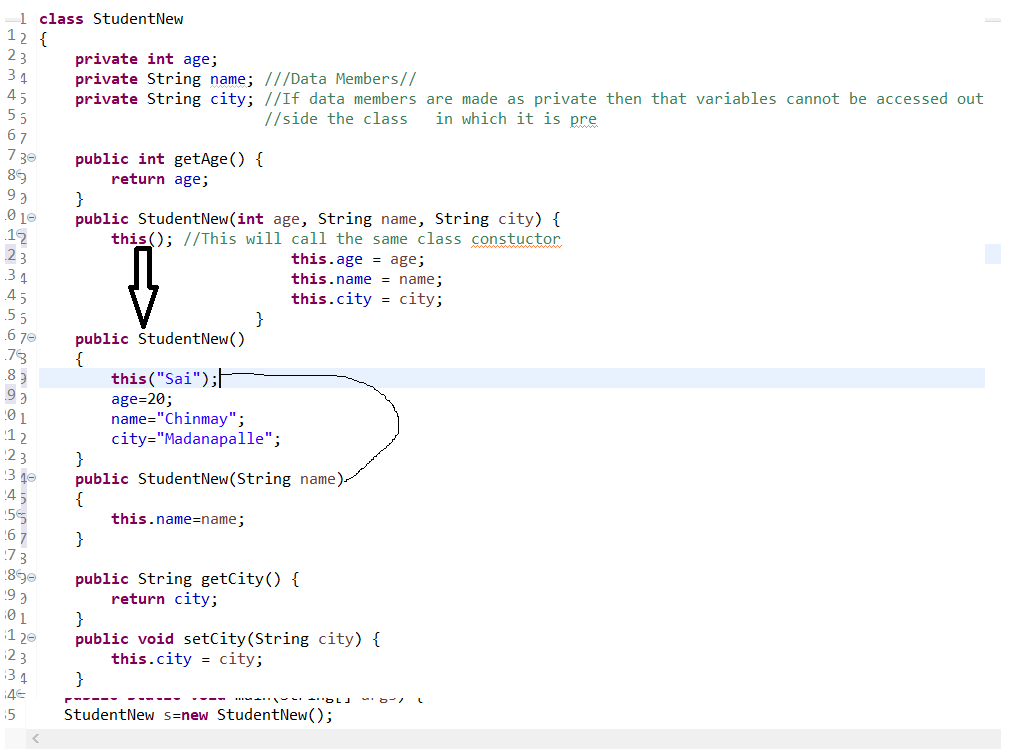
name="Chinmay";

city="Madanapalle";

}

Constructor can either this()/super() as the first line of constructor.

Constructor Chaining: Constructor chaining is the process of calling one constructor from another constructor .



There are two ways.

* **Within same class**: It can be done using **this()** keyword for constructors in the same class
* **From base class:**by using **super()** keyword to call the constructor from the base class.

**Rules of constructor chaining :**

1. The **this()** expression should always be the first line of the constructor.
2. There should be at-least be one constructor without the this() keyword (constructor 3 in above example).Other wise it will lead to stack overflow(One constructor calling another Constructor)
3. Constructor chaining can be achieved in any order.

**Constructor Chaining to other class using super() keyword :**

// Java program to illustrate Constructor Chaining to

// other class using super() keyword

**class** Base

{

String name;

// constructor 1

Base()

{

**this**("");

System.***out***.println("No-argument constructor of" +

" base class");

}

// constructor 2

Base(String name)

{

**this**.name = name;

System.***out***.println("Calling parameterized constructor"

+ " of base");

}

}

**class** Derived **extends** Base

{

// constructor 3

Derived()

{

System.***out***.println("No-argument constructor " +

"of derived");

}

// parameterized constructor 4

Derived(String name)

{

// invokes base class constructor 2

**super**(name);

System.***out***.println("Calling parameterized " +

"constructor of derived");

}

**public** **static** **void** main(String args[])

{

// calls parameterized constructor 4

Derived obj = **new** Derived("test");

// Calls No-argument constructor

// Derived obj = new Derived();

}

}

Output :

Calling parameterized constructor of base

Calling parameterized constructor of derived

When we want certain common resources to be executed with every constructor we can put the code in the **init block**. Init block is always executed before any constructor, whenever a constructor is used for creating a new object.

**class** Temp

{

// block to be executed first

{

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

}

Temp()

{

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

}

Temp(**int** x)

{

System.***out***.println(x);

}

// block to be executed after the first block

// which has been defined above.

{

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

}

**public** **static** **void** main(String args[])

{

**new** Temp();

**new** Temp(10);

}

}

A screenshot of a computer program

Description automatically generated

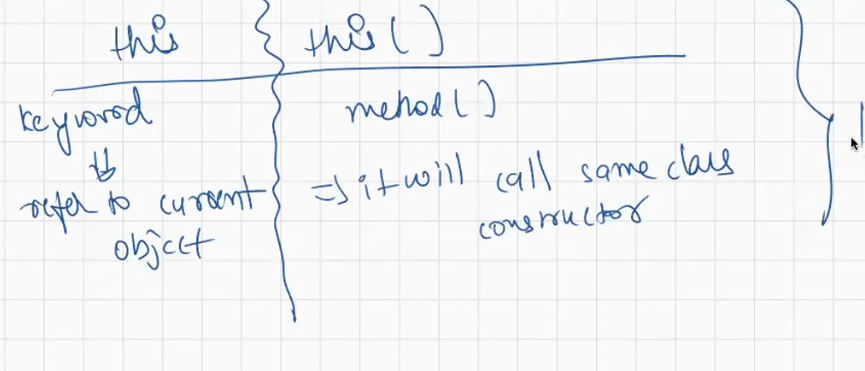
Init blocks will get executed whenever a new Object gets created.

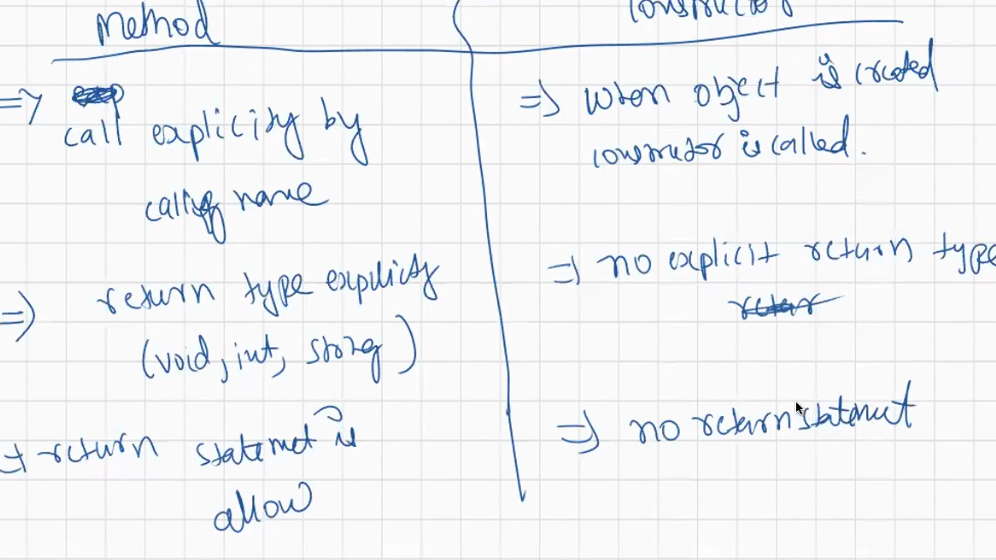
A close-up of a paper

Description automatically generated

We cannot have both and this()/super() in the same constructor.

If there are something that needs to be executed during Object creation then we can keep that in the constructor.





When we want something to executed during the Object creation we can specify them in the

Constructor.

**class** DB

{

DB()

{

System.***out***.println("Connected to Database Success");

}

}

**public** **class** DBConnection {

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

// **TODO** Auto-generated method stub

DB sb=**new** DB();

}

}