What is constructor Overloading :

Defining **multiple constructors** in the **same class [Method Overloading is possible in super & sub class but constructor overloading is only possible in the same class]** but differ in :
1) No of Parameters
2) Type of Parameters
3) Order of parameter

THE FIRST LINE OF ANY CONSTRUCTOR IS **RESERVED FOR EITHER** super() [super off OR this() [this off]
If we write super() OR this() in the first line of constructor [either super() OR this() then automatically java compiler will add super() as the first line of constructor.

Case 1:

Demo.java

```
public class Demo
{
}
```

javac

Demo.class

```
public class Demo
{
  public Demo() //added by java compiler
  {
    super(); //added by java compiler
  }
}
```

Case 2 :

Calculate.java

```
class Calculate
{
    public Calculate(int x, int y)
    {
        this(2.5, 6.7);
    }

    public Calculate(double u, double v)
    {
    }
}
```

javac

Calculate.class

```
class Calculate
{
  public Calculate(int x, int y)
  {
    this(2.5, 6.7);
  }

  public Calculate(double u, double v)
  {
    super(); //added by compiler
  }
}
```

* this() is used to call the current class constructor.

* From Java 22V onwards super() OR this() we can write anywhere in the constructor body.

* While working with constructor overloading, using single object we can call multiple constructors but compulsorily we need to use this() with appropriate arguments.

```
package com.ravi.constructor_overloading;

public class ConstructorOverloading
{
  void main()
  {
    new Calculate(10, 20);
  }

  class Calculate
  {
    public Calculate(int x, int y)
    {
      this(2.5, 6.7);
      System.out.println("Sum of two integer is :"+(x+y)); //1rd Output
    }

    public Calculate(double u, double v)
    {
      System.out.println("Sum of two double is :"+(u+v));  //1st output
      this("Data", "Base"); //from Java 22V onwards
    }

    public Calculate(String s, String y)
    {
      System.out.println("Concatenation of two String is :"+(s+y)); //2nd Output
    }
  }
}
```

Modifiers applicable for Constructor :

* We can apply all the access modifiers on the constructor like private, default (private-package), protected & public.

* A constructor we cannot declare with **Final, static, abstract and synchronized modifier.**

Q1 :
Why we should declare a constructor with private access modifier.

We should declare a constructor with private access modifier due to following two reasons :

1) If we want to declare only static field & static method inside that particular class then we should declare a constructor with private access modifier so outsider cannot create the object for that particular class.
Example :
Java.lang.Math class & java.util.Arrays class are the best example

2) To achieve **singlton design** pattern.

```
package com.ravi.constructor_overloading;

class Test
{
  public static Test t = null;

  private Test()
  {
    System.out.println("No Argument Constructor");
  }

  public static Test getObject()
  {
    t = new Test();
    return t;
  }

  public void greet()
  {
    System.out.println("Hello Everyone from Batch 32");
  }

}

public class SinglTonDesignPattern
{
  public static void main(String[] args)
  {
    Test test = Test.getObject();
    test.greet();
  }

}
```

What is non static block OR Instance Initiator :

* A **non static block** is a special block which will be **automatically** executed at the time of creating the object.
Example :

```
class Test
{
  {
    //non static block
  }
}
```

* Every time we will create an object in Java then non static block will be executed.

NSB1.java

```
package com.ravi.non_static_block;

class Test
{
    {
      System.out.println("Non static Block");
    }
}

public class NonStaticBlockDemo
{
  public static void main(String[] args)
  {
    new Test();  //Nameless Object OR Anonymous object
    new Test();  //Nameless Object OR Anonymous object
  }
}
```

* The 2nd line of **constructor body will be reserved for non static block** (If available) that means java compiler will automatically add non static block in the 2nd line of constructor (If and only if, the first line of constructor contains super()).

Case 1 :

Test.java

```
public class Test
{
  public Test()
  {
    System.out.println("Constr");
  }

  {
    System.out.println("NSB");
  }
}
```

javac

Test.class

```
public class Test
{
  public Test()
  {
    super(); //1st line is super
    {
      //NSB
      System.out.println("NSB");
    }

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

Case 2 :

Calculate.java

```
class Calculate
{
  public Calculate(int x, int y)
  {
    this(2.5, 6.7);
  }

  public Calculate(double u, double v)
  {
  }

  {
    System.out.println("NSB");
  }
}
```

javac

Calculate.class

```
class Calculate
{
  public Calculate(int x, int y)
  {
    this(2.5, 6.7);
  }

  public Calculate(double u, double v)
  {
    super();
    {
      System.out.println("NSB");
    }
  }
}
```

SO CONCLUSION IS, THE CONSTRUCTOR WHICH CONTAINS super() in the first line only those constructor will contain non static block in the 2nd line.