--We can overload the main method as well ,

example:

```
package com.masai;

public class Demo {


	public static void main(int i) {

		System.out.println("inside main(int i) of Demo");

	}



	public static void main(String[] args) {


		main(10);
		Demo.main(20);
		new Demo().main(30);

	}

}
```

--we can call the main method of another class inside another class also.

A.java:

```
package com.masai;

public class A {

	int i = 10;

	void funA() {

		System.out.println("inside funA of A");
	}
```

```
	public static void main(String[] args) {
		System.out.println("inside main of A");
	}

}
```

Demo.java:

```
---------------

package com.masai;

public class Demo {


	public static void main(String[] args) {


		A.main(null);

	}

}
```

Constructor in java:

==========

--it is a kind of non-static method which will be executed automatically at the time of creating an object.

ex:

```
Demo d1= new Demo();
```

--the meaning of the above statement is "creating a Demo class obj by calling /executing zero argument constructor of Demo class".

Note: when we compile a .java file of a class, Java compiler verifies, is there any constructor in our .java file or not. if we place any constructor manually inside a .java file,

Java compiler will place the same constructor inside the .class file , but if we don't place any constructor explicitly inside our .java file then java compiler will place a default constructor inside the .class file.

ex:

```
class Demo{

}
```

java compiler
Demo.java ----------> Demo.class

Note: - we can have a .java file for a class without a constructor, but we can't have a .class file for a class without a constructor.

--default constrcutor given by the java compiler will always public and zero argument and it is empty body.

ex: compiler given default constructor

```
public Demo(){
//super();
}
```

--strikly speaking , it is not a empty body , there is one hidden statement is there inside the default constructor as a first statement.

--until the last statement of the constructor is not executed, object is not created completly. so for an object creatiion, constructor execution is mandatory.

Diff bt normal method and constructor:
==========

Normal method	Constructor
1. method name can be any name.	1. constructor name must be the class name
2. a method must have a return type atleast void	2. constructor does not have return type

3.method can be static **3. static keyword is not applicable with constructor**

4.on a single obj we can call a method multiple time. **4. on a single obj a constructor will be called only one**

5.a method can be abstract and can be final also **5. abstract and final keyword is not applicable with constructor.**

Similarities
=========

1. both are the code block, we can write multiple executable statements.
2.as we can overload a method, we can overload a constructor also. and all the static polymorphism rules are applicable with the constructor overloading.

constructor overloading:
==========

ex:

Demo.java:-

```
package com.masai;

public class Demo {

	int x = 10;
	int y;

	void fun1() {
		System.out.println("inside fun1 of Demo");
	}


	Demo(){
		System.out.println("inside Demo()....");
	}

	Demo(int i){
		System.out.println("inside Demo(int )....");
		System.out.println(i);
	}
```

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

		Demo d1 = new Demo();
		Demo d2 = new Demo(10);
		//Demo d2 = new Demo("hello");//CE


	}

}
```

--if we place 4 overloaded constructor inside our class, then we can create object of our class in 4 ways.

this keyword:
==========

--it will represent the current class object.

--there are 3 uses of 'this' keyword:

1. to represent the current class object.

2.to defrentiate the instance variable and the local variable

3. to call a constructor of a class from the another constructor of the same class.

Example:

Demo.java:

```
package com.masai;

public class Demo {

	int x = 10;
```

```
	void fun1() {
		int x=400;
		System.out.println("inside fun1 of Demo");
		System.out.println(x);//local variable
		System.out.println(this.x); // instance variable

		System.out.println(this);//current object on which fun1 is called


	}


	public static void main(String[] args) {

		Demo d1= new Demo();
		System.out.println(d1);
		d1.fun1();

	}

}
```

***Note: 'this' keyword we can not use inside static area.

Demo.java:-

```
---------- 

package com.masai;

public class Demo {

	int x = 10;

	public static void main(String[] args) {

		int x=200;

		System.out.println(this.x);


	}
```

}

Note: constructor will be called automatically whenever we create obj of a cass, but we can also call a constructor explicitly.

--if we want to call a constructor, then that call must be from the another constructor of the same class (by using 'this' keyword) or from the constructor of child class (by using 'super' keyword)

--that call of the constructor must be the first statement inside a constructor.

example:

Demo.java:-

```
----------
package com.masai;

public class Demo {


	Demo(){
		this(10);
		System.out.println("inside Demo()....");
	}

	Demo(int i){
		this("hello");
		System.out.println("inside Demo(int )....");
		System.out.println(i);
	}

	Demo(String s){

		System.out.println("inside Demo(String )....");
		System.out.println(s);
	}



	public static void main(String[] args) {

		Demo d1=new Demo();
```

```
    }

}
```

Constructor are used for basically for 2 purpose :

1. if we want to execute some statement at the time of our object creation, then we can keep those statements inside the constructor.

2. to initialize the instance variable (initialize an object).

example:

Student.java:

```
---------------

package com.masai;

public class Student {

    int roll;
    String name;
    int marks;


    //zero argument constructor
    Student(){

    }

    //parameterized constructor
    Student(int roll, String name, int marks){
        this.roll=roll;
        this.name=name;
        this.marks=marks;
    }



    public void showDetails() {
```

```
                System.out.println("Roll is :"+roll);
                System.out.println("Name is :"+name);
                System.out.println("Marks is :"+marks);
        }

}
```

Demo.java:

```
package com.masai;

public class Demo {


        public static void main(String[] args) {

                Student s1= new Student(10,"Ram",780);

                Student s2=new Student();
                s2.roll=100;
                s2.name="Ravi";
                s2.marks=900;


                s1.showDetails();
                s2.showDetails();


        }

}
```

pure encapsulation:
==========

--mark our class variable as private and expose them outside the class through the public getters and setters method.

--don't expose our data directly.

Java Bean class:
==========
--it is a reusable, universal component which should have following properties:

1.this class should be public

2. variables/fields should be private

3.for each variable/field there should be corresponding public getter and setter methods.

4.this class must have zero argument constructor/default constructor

5.this class may have parameterized constructor. (optional)

Student.java: as a Bean class

```
package com.masai;

public class Student {

	private int roll;
	private String name;
	private int marks;


	public Student() {

	}


	public Student(int roll, String name, int marks) {
		super();
		this.roll = roll;
		this.name = name;
		this.marks = marks;
	}
```

```
	public int getRoll() {
		return roll;
	}
	public void setRoll(int roll) {
		this.roll = roll;
	}


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

	public int getMarks() {
		return marks;
	}

	public void setMarks(int marks) {
		this.marks = marks;
	}

}
```

Demo.java:

```
package com.masai;

public class Demo {


	public static void main(String[] args) {


		Student s1= new Student(10, "Ram", 780);

		Student s2= new Student();
		s2.setRoll(20);
		s2.setName("Ramesh");
		s2.setMarks(780);

		System.out.println("Roll is :"+s1.getRoll());
```

```
                System.out.println("Name is :"+s1.getName());
                System.out.println("Marks is :"+s1.getMarks());

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

                System.out.println("Roll is :"+s2.getRoll());
                System.out.println("Name is :"+s2.getName());
                System.out.println("Marks is :"+s2.getMarks());


        }

}
```