Constructor

It is a member method which has the same name as the class name.
How much space will be taken by an Object will be given by Constructor.
Even if we don't define a constructor, there will be a default constructor for every class.
Public A()→Constructor

- It is a Member Method
- Same Name as Class Name
- Constructor will never return anything [No Return type required].
- It will be used to allocate memory
- It can be also used for initializing your value

Note:

- We cannot call a constructor explicitly like methods.
- Constructor gets called when an object is created

Type of Constructor:

1. **Default**
2. **Parameterized**

Note: Once you define a parameterized constructor and then creating object with Default Constructor, then JVM will not by default provide Default Constructor, that means you must also define Default Constructor first, Otherwise it will throw error.

Example:

```
public class Parameterized_Constructor {

	public static void main(String[] args) {
	A a=new A();// This line will throw compilation error// It will tell you to define
			the default constructor or use the parameterized constructor
	}
}

class A {
	public A(int i) {
		System.out.println(i);
	}
   }
```

So we have to define here the default Constructor

```
   public A() {
// TODO Auto-generated constructor stub
	}
```

Q] Can we have two constructors with same name?

Ans: Yes we can have two constructors with same name, but with different Parameters.
Even more than two Constructors is also possible.
And this concept is called as **Constructor Overloading**

```
public class Parameterized_Constructor {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		A a=new A();
	}
}

class A {

	public A() {
	System.out.println(" In A Constructor");  //Output will be: In A Constructor
	}
	public A(int i) {
		System.out.println(" In A Constructor para");
	}

	public A(int i,double d) {
		System.out.println(" In A Constructor intDouble");
	}
   }
```

Constructor Overloading Implicit Conversion:

Generally We See

```
class A {

	public A() {
	System.out.println(" In A Constructor");
	}
	public A(int i) {
		System.out.println(" In A Constructor int");
	}

	public A(double d) {
		System.out.println(" In A Constructor Double");
	}
}
```

In the above example,

- If we call A(), default constructor gets called.
- If we call A(5), then output `In A Constructor int`
- If we call A(5.5), then output `In A Constructor Double`

Now, if A has only two constructors,

```
class A {

	public A() {
	System.out.println(" In A Constructor");
	}


	public A(double d) {
		System.out.println(" In A Constructor Double");
	}
}
```

And `A a=new A(5);`

But it does not have the int constructor, so it will implicitly convert int to double but the vice-versa will throw error

Output: `In A Constructor Double`