Constructor :

- Whenever we are creating an object some piece of the code will be executed automatically to perform initialization of an object this piece of code is nothing but a constructor.
- Main objective of the constructor is nothing but initialisation of Object.
- Constructor can be referred to as a specialised setter whose name is the same that of the class name and it doesn't have any return type.
- Constructor is invoked during object creation.

Rules for writing a constructor
===============================
1. Name of the constructor and name of the class must be same.
2. Return type concept not applicable for constructor, even if we provide it won't result in compile time error, if we do so then java language will treat this as "normal method".

```
    eg:: class Test{
                 void Test(){
                          System.out.println("Hello");// It is not a constructor,it is a
method.
                 }
        }
```

3. It is not a good practise to take the method name same as that of the classname.
4. The modifiers applicable for constructors are private,public,protected,default.
5. The other modifiers if we use, it would result in compile time error.

```
      eg:: class Test{
                 static Test(){
            }
            }
         output::CE
```

Default constructor
===================
1. For every java class constructor concept is applicable.
2. If we don't write any constructor, then compiler will generate default constructor.
3. If we write atleast one constructor then compiler won't generate any default constructor, so
   we say every java class will have compiler generated default constructor or programmer written
   constructor but not both simultaneously.

Prototype of default constructor
================================
1. It is always no argument constructor.

2. The access modifier of the default constructor is same as class modifier.
   [applicable for public and default]
3. Default constructor contains one line, super(), It is a call to super class constructor.
   eg:: constructor.png

super() vs this()
=================
1. The first line inside the constructor can be super()/ this().
2. If we are not writing anything then compiler will generate super();

case1: We have to take super()/this() only in the first line of constructor, if we are writing any where else it would result in compile time error.
eg#1::
```
  class Test{
          Test(){
                    System.out.println("Constructor");//CE
                    super();
          }
  }
```

eg#2::we can either use super()/this() but not simulatenously
```
 class Test{
          Test(){
               super();
               this();//CE
          }
 }
```

eg#3:: we can use super()/this() only inside the constructor otherwise it would result in
   compile time error.

```
  class Test{
          void methodOne(){
                    super();
                    this();
          }
  }
```

Note::
super() => It should be first line in the constructor.
   It should be used only in constructor.
   It will take the control to parent class constructor.

this() => It should be first line in the constructor.
   It should be used only in constructor.

It will make the call of current class constructor.

Difference b/w super(),this() and super,this?
super(),this()

1. These are constructor calls
2. These are used to invoke super class and current class constructor directly
3. We should use only inside the constructor that to first line otherwise we get compile time error.

super, this
1. These are reservered words meant while working with object creation
2. These are used to refer to parent class and child class instance members.
3. We can use anywhere(instance area),except static area otherwise we get compile time error.

```
eg:: class Test{
            public static void main(String... args){
                    System.out.println(super.hashCode());//CE
            }
      }
```

Constructor Overloading
=====================

- A class can contain more than one constructor and all these constructors have same name they differ only in the type of argument, hence these constructors are considered as "Overloaded constructor".

eg::
```
class Test {
        Test(double d) {
                System.out.println("double argument constructor");
        }

        Test(int i) {
                this(10.5);
                System.out.println("int argument constructor");
        }

        Test() {
                this(10);
                System.out.println("no argument constructor");
        }
}
public class MainApp {
        public static void main(String[] args) throws Exception {
                Test t1= new Test();//double int no argument constructor
                Test t2= new Test(10);// double int argument constructor
```

```
                    Test t3= new Test(10.5);//dobule argument constructor
          }
}
```