- constructor is a block of codes similar to the method.
- It is called when an instance of the class is created.
- At the time of calling constructor, memory for the object is allocated in the memory
- Every time an object is created using the new() keyword, at least one constructor is called.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
- .

Rules for creating Java constructor

- There are two rules defined for the constructor.
- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

- Default constructor (no-arg constructor)
- Parameterized constructor
-

  A constructor is called "Default Constructor" when it doesn't have any parameter.

```java
class Bike1{
Bike1(){System.out.println("Bike is created");}

public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

```
class Student3{
int id;
String name;

void display(){
System.out.println(id+" "+name);}

public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();

s1.display();
s2.display();
}
}
```

- A constructor which has a specific number of parameters is called a parameterized constructor.
- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

```
class Student4{
  int id;
  String name;
    Student4(int i,String n){
  id = i;
  name = n;
  }
   void display(){System.out.println(id+"  "+name);}

  public static void main(String args[]){
   Student4 s1 = new Student4(111,"Karan");
  Student4 s2 = new Student4(222,"Aryan");
   s1.display();
  s2.display();
 }
}
```

Java Private No-arg Constructor

```
class Main {

  int i;

private Main() {
    i = 5;
    System.out.println("Constructor is called");
  }

  public static void main(String[] args) {

Main obj = new Main();
    System.out.println("Value of i: " + obj.i);
  }
}
```

```
class Main9 {

  int i;

private Main9() {
    i = 5;
    System.out.println("Constructor is called");
  }

  public static void main(String[] args) {

Main9 obj = new Main9();
    System.out.println("Value of i: " + obj.i);
  }
}
```

Parameterized Constructor

```
class Main {

  String languages;

Main(String lang) {
    languages = lang;
    System.out.println(languages + " Programming Language");
  }

  public static void main(String[] args) {

Main obj1 = new Main("Java");
    Main obj2 = new Main("Python");
    Main obj3 = new Main("C");
  }
}
```

Default Constructor

```
class Main {

  int a;
  boolean b;

  public static void main(String[] args) {

Main obj = new Main();

    System.out.println("Default Value:");
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
  }
}
```

```
class Student8 {
  String firstName;
  String lastName;
  int age;
    public static void main(String args[]) {
      Student8 myStudent = new Student8();
            myStudent.firstName = "Ihechikara";
      myStudent.lastName = "Abba";
      myStudent.age = 100;
            System.out.println(myStudent.age);

      System.out.println(myStudent.firstName);


  }
}
```

Constructor Overloading

```
class Student5{
    int id;
    String name;
    int age;
     Student5(int i,String n){
    id = i;
    name = n;
    }
       Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
void display()
{System.out.println(id+" "+name+" "+age);}
      public static void main(String args[]){
   Student5 s1 = new Student5(111,"Karan");
   Student5 s2 = new Student5(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

```
class Main {
  String language;
Main() {
    this.language = "Java";
  }
Main(String language) {
    this.language = language;
  }
```

```
 public void getName() {
    System.out.println("Programming Language: " + this.language);
  }
  public static void main(String[] args) {
Main obj1 = new Main();
Main obj2 = new Main("Python");
    obj1.getName();
    obj2.getName();
  }
}
```

this keyword is used to refer to the current object inside a method or a constructor.

•Constructors are invoked implicitly when you instantiate objects.
•The two rules for creating a constructor are:
1. The name of the constructor should be the same as the class.
2. A Java constructor must not have a return type.
•If a class doesn't have a constructor, the Java compiler automatically creates a **default constructor** during run-time.
•The default constructor initializes instance variables with default values. For example, the int variable will be initialized to 0
•Constructor types:
No-Arg Constructor - a constructor that does not accept any arguments
Parameterized constructor - a constructor that accepts arguments
Default Constructor - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
•A constructor cannot be abstract or static or final.
•A constructor can be overloaded but can not be overridden.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

WAP to count the number of objects made of a particular class using static variable and static method and display the same.

```
class ObjectCounter {
    static  int count = 0; // Static variable to store the count
    //  ObjectCounter() {
      /// count++; // Increment count every time an object is created
  // }
      static int getCount() {
count++;
      return count; // Static method to return the count
   }
public static void main(String[] args) {
    // Creating some objects of ObjectCounter class
    ObjectCounter obj1 = new ObjectCounter();
    System.out.println("Number of objects created: " + getCount());

    ObjectCounter obj2 = new ObjectCounter();
    System.out.println("Number of objects created: " + getCount());

    ObjectCounter obj3 = new ObjectCounter();
    System.out.println("Number of objects created: " + getCount());



    // Displaying the count of objects created
    //System.out.println("Number of objects created: " + getCount());
  }
}
```

WAOOP to count the no. of objects created of a class using constructors.

```
class ObjectCounter {
    static  int count = 0; // Static variable to store the count
    ObjectCounter() {
      count++; // Increment count every time an object is created
   }
      static int getCount() {
return count; // Static method to return the count
   }
public static void main(String[] args) {
        // Creating some objects of ObjectCounter class
        ObjectCounter obj1 = new ObjectCounter();
        System.out.println("Number of objects created: " + getCount());

        ObjectCounter obj2 = new ObjectCounter();
        System.out.println("Number of objects created: " + getCount());

        ObjectCounter obj3 = new ObjectCounter();
        System.out.println("Number of objects created: " + getCount());



    }
}
```

```
class Comp1 {
    private int a, b;
        Comp1( int i,  int j ){
        this.a = i;
        this.b = j;
    }
Comp1(int i){
               this(i, i);
    }
Comp1(){
                this(0);
    }
       public String toString(){
        return this.a + " + " + this.b + "i";
    }
public static void main( String[] args )
{
            Comp1 c1 = new Comp1(2, 3);
    Comp1 c2 = new Comp1(3);
    Comp1 c3 = new Comp1();
    System.out.println(c1);
    System.out.println(c2);
    System.out.println(c3);
  }
}
```