**Constructors in Java**

Constructors are used to initialize a newly created object.to remove the garbage value from an object

Constructor don’t have return type

Name of the constructor and class name are same.

We cannot use static, abstract, final keyword with a constructor

But we can pass a private constructor. Which means we can restrict the class to create an object. In that case we have to declared all the methods of the class as static so that we can call them without creating the object of the class.

In the object initialization the passing argument should maintain the same ordering as parameter we pass

We use this keyword to refer the current object

There are three types of constructors such as

1. Default constructor
2. No-argument constructor
3. Parameterize constructor

If we do not pass any constructor in our program then compiler pass a default constructor in our program. Which is not visible in our code.it is executed during the execution of the program.

The constructor which doesn’t contain any argument are called no-arg constructor.however it can have body

class Demo

{

public Demo()

{

System.out.println("This is a no argument constructor");

}

public static void main(String args[]) {

new Demo();

}

}

Constructor with arguments (or you can say parameters) is known as [Parameterized constructor](https://beginnersbook.com/2014/01/parameterized-constructor-in-java-example/).

public class Employee {

int empId;

String empName;

Employee (int id, String name) {

this.empId = id;

this.empName = name;

}

}

### **Constructor Chaining**

Constructor chaining is a process of calling one constructor from another constructor in the same class. Since constructor can only be called from another constructor, constructor chaining is used for this purpose.

To call constructor from another constructor this keyword is used. This keyword is used to refer current object.

 The real purpose of Constructor Chaining is that you can pass parameters through a bunch of different constructors, but only have the initialization done in a single place. This allows you to maintain your initializations from a single location, while providing multiple constructors to the user

class Employee

{

public String empName;

public int empSalary;

public String address;

//default constructor of the class

public Employee()

{

//this will call the constructor with String param

this("Chaitanya");

}

public Employee(String name)

{

//call the constructor with (String, int) param

this(name, 120035);

}

public Employee(String name, int sal)

{

//call the constructor with (String, int, String) param

this(name, sal, "Gurgaon");

}

public Employee(String name, int sal, String addr)

{

this.empName=name;

this.empSalary=sal;

this.address=addr;

}

void disp() {

System.out.println("Employee Name: "+empName);

System.out.println("Employee Salary: "+empSalary);

System.out.println("Employee Address: "+address);

}

public static void main(String[] args)

{

Employee obj = new Employee();

obj.disp();

}

}

## Constructor Overloading

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.

class StudentData

{

private int stuID;

private String stuName;

private int stuAge;

StudentData()

{

//Default constructor

stuID = 100;

stuName = "New Student";

stuAge = 18;

}

StudentData(int num1, String str, int num2)

{

//Parameterized constructor

stuID = num1;

stuName = str;

stuAge = num2;

}

//Getter and setter methods

public int getStuID() {

return stuID;

}

public void setStuID(int stuID) {

this.stuID = stuID;

}

public String getStuName() {

return stuName;

}

public void setStuName(String stuName) {

this.stuName = stuName;

}

public int getStuAge() {

return stuAge;

}

public void setStuAge(int stuAge) {

this.stuAge = stuAge;

}

public static void main(String args[])

{

StudentData myobj = new StudentData();

System.out.println("Student Name is: "+myobj.getStuName());

System.out.println("Student Age is: "+myobj.getStuAge());

System.out.println("Student ID is: "+myobj.getStuID());

StudentData myobj2 = new StudentData(555, "Chaitanya", 25);

System.out.println("Student Name is: "+myobj2.getStuName());

System.out.println("Student Age is: "+myobj2.getStuAge());

System.out.println("Student ID is: "+myobj2.getStuID());

}

}

## Java Copy Constructor

A copy constructor is used for copying the values of one object to another object

class JavaExample{

String web;

JavaExample(String w){

web = w;

}

JavaExample(JavaExample je){

web = je.web;

}

void disp(){

System.out.println("Website: "+web);

}

public static void main(String args[]){

JavaExample obj1 = new JavaExample("BeginnersBook");

JavaExample obj2 = new JavaExample(obj1);

obj1.disp();

obj2.disp();

}

}

1. Every class has a constructor whether it’s a normal class or an abstract class.
2. Constructors are not methods and they don’t have any return type.
3. Constructor name should match with class name.
4. Constructor can use any access specifier; they can be declared as private also. Private constructors are possible in java but their scope is within the class only.
5. Like constructors’ method can also have name same as class name, but still, they have return type, though which we can identify them that they are methods not constructors.
6. If you don’t implement any constructor within the class, compiler will do it for.
7. this () and super () should be the first statement in the constructor code. If you don’t mention them, compiler does it for you accordingly.
8. Constructor overloading is possible but overriding is not possible. Which means we can have overloaded constructor in our class but we can’t override a constructor.
9. Constructors cannot be inherited.
10. If Super class doesn’t have a no-arg(default) constructor then compiler would not insert a default constructor in child class as it does in normal scenario.
11. Interfaces [do not have constructors](https://beginnersbook.com/2013/12/java-constructor-in-interface/).
12. Abstract class can have constructor and it gets invoked when a class, which implements interface, is instantiated. (i.e. object creation of concrete class).
13. A constructor can also invoke another constructor of the same class – By using this(). If you want to invoke a parameterized constructor then do it like this: this(parameter list).
14. There is no static constructor in java