Constructor Overloading

Like methods, **constructors** can also be overloaded.

Let's understand what is constructor overloading and Why do we do it by considering an example shown below:

Example:

```
class Car
{
    private String name;
    private int mileage;
    private int cost;
    public Car()
    {
        name = "BMW";
        mileage = 10;
        cost = 7000000;
    }
    public Car(String name,int mileage,int cost)
    {
        this.name = name;
        this.mileage = mileage;
        this.cost = cost;
    }
    public String getName()
    {
        return name;
    }
    public int getMileage()
    {
         return mileage;
    }
    public int getCost()
    {
         return cost;
    }
}
```

Zero parameterized constructor

Parameterized constructor

```
class Demo
{
    public static void main(String[] args)
    {
        Car c1 = new Car();
        System.out.println(c1.getName());
        System.out.println(c1.getMileage());
        System.out.println(c1.getCost());

        Car c2 = new Car("Ferrari",5,9000000);
        System.out.println(c2.getName());
        System.out.println(c2.getMileage());
        System.out.println(c2.getCost());
    }
}
```

Output:

Ferrari
5
9000000

What is constructor overloading?

Having Multiple constructors within a class is referred to as constructor overloading.

Why do we do it?

Constructor overloading is required so that different objects can be initialized differently.

In one year, Java gets downloaded one billion times.

Local Chaining

What is local chaining?

Local Chaining is the process of a constructor of a class calling another constructor of the same class.

Why do we do it?

Local chaining allows you to maintain your initialization from a single location, while providing multiple constructors to the user.

How to achieve local chaining?

*Local chaining can be achieved using **this()** function call.*
*this() should compulsorily be the **first line** in the constructor.*

Let's understand what is local chaining by considering the example shown below:

Example:

```
class Car
{
    private String name;
    private int mileage;
    private int cost;
    public Car()
    {
        name = "BMW";
        mileage = 10;
        cost = 7000000;
    }
    public Car(String name,int mileage,int cost)
    {
        this();
    }
```

```
    public String getName()
    {
        return name;
    }
    public int getMileage()
    {
         return mileage;
    }
    public int getCost()
    {
         return cost;
    }
}

class Demo
{
    public static void main(String[] args)
    {
        Car c1 = new Car("Ferrari",5,9000000);
        System.out.println(c1.getName());
        System.out.println(c1.getMileage());
        System.out.println(c1.getCost());
    }
}
```

In the above example,

```
this.name = name;
this.mileage = mileage
this.cost = cost;
```

Replaced by this()
In parametrized constructor

In the above code, during object creation we are calling 3 parameterized constructor inside which first line is this() which will now give control to zero parameterized constructor and this way we achieve local chaining.

Let us understand this() in detail by considering different examples.

Example - 1

```
class Car
{
     private String name;
     private int mileage;
     private int cost;
     public Car()
     {
          name = "BMW";
          mileage = 10;
          cost = 7000000;
     }
     public Car(String name,int mileage,int cost)
     {
          this();
          this.name = name;
          this.mileage = mileage;
          this.cost = cost;
     }
     public String getName()
     {
          return name;
     }
     public int getMileage()
     {
           return mileage;
     }
     public int getCost()
     {
           return cost;
     }
}
class Demo
{
    public static void main(String[] args)
    {
        Car c1 = new Car("Ferrari",5,9000000);
        System.out.println(c1.getName());
        System.out.println(c1.getMileage());
        System.out.println(c1.getCost());
    }
}
```

In the above code, during object creation we are calling 3 parameterized constructor inside which the first line is this() which will now control to zero parameterized constructor. Once the body of zero parameterized constructor execute, control comes back to where it came from which is parameterized constructor.

Example – 2

```
class Car
{
    private String name;
    private int mileage;
    private int cost;
    public Car()
    {
        name = "BMW";
        mileage = 10;
        cost = 7000000;
    }
    public Car(String name,int mileage,int cost)
    {
        this(name);
    }
    public Car(String name)
    {
        this();
        this.name = name;
    }

   public String getName()
   {
       return name;
   }
   public int getMileage()
   {
        return mileage;
   }
   public int getCost()
   {
        return cost;
   }
}
```

```java
class Demo
{
    public static void main(String[] args)
    {
        Car c1 = new Car("Ferrari",5,9000000);
        System.out.println(c1.getName());
        System.out.println(c1.getMileage());
        System.out.println(c1.getCost());
    }
}
```

In the above code, during object creation we are calling 3 parameterized constructor. Inside 3 parameterized constructor, the first line is this(name) which will now give control to one parameterized constructor inside which first line is this() which in turn will give control to 0 parametrized constructor. In this way, chaining between different constructor takes place using this().

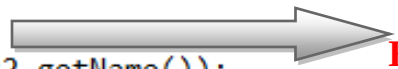
He got it. Did you?

Try tracing the codes given below:

Example – 3

```
class Car
{
    private String name;
    private int mileage;
    private int cost;
    public Car(String name,int mileage,int cost)
    {
        this.name = name;
        this.mileage = mileage;
        this.cost = cost;
    }
    public String getName()
    {
        return name;
    }
    public int getMileage()
    {
         return mileage;
    }
    public int getCost()
    {
         return cost;
    }
}
class Demo
{
    public static void main(String[] args)
    {
        Car c1 = new Car("Ferrari",5,9000000);
        System.out.println(c1.getName());
        System.out.println(c1.getMileage());
        System.out.println(c1.getCost());

        Car c2 = new Car();
        System.out.println(c2.getName());
        System.out.println(c2.getMileage());
        System.out.println(c2.getCost());
    }
}
```



Output: **Error (there is no zero parameterized constructor in your code)**

The error in the above code is removed by inserting a zero parameterized constructor as shown below.

Example - 4

```
class Car
{
    private String name;
    private int mileage;
    private int cost;
    public Car(String name,int mileage,int cost)
    {
        this.name = name;
        this.mileage = mileage;
        this.cost = cost;
    }
    public Car()
    {

    }
    public String getName()
    {
        return name;
    }
    public int getMileage()
    {
         return mileage;
    }
    public int getCost()
    {
         return cost;
    }
}
class Demo
{
    public static void main(String[] args)
    {
        Car c1 = new Car("Ferrari",5,9000000);
        System.out.println(c1.getName());
        System.out.println(c1.getMileage());
        System.out.println(c1.getCost());

        Car c2 = new Car();
        System.out.println(c2.getName());
        System.out.println(c2.getMileage());
        System.out.println(c2.getCost());
    }
}
```