JAVA Constructor

28 October 2025 18:47

A constructor in java is a special method that is used to initialize object when they are created. It is automatically called when an object is instantiated using the new keyword.
Its primary purpose is to initialize the newly created object.

Characteristics of constructor

Same name as the class:- A constructor has the same name as the class in which it is defined.

```
class Student {
    Student() { } // Constructor name = Class name
}
```

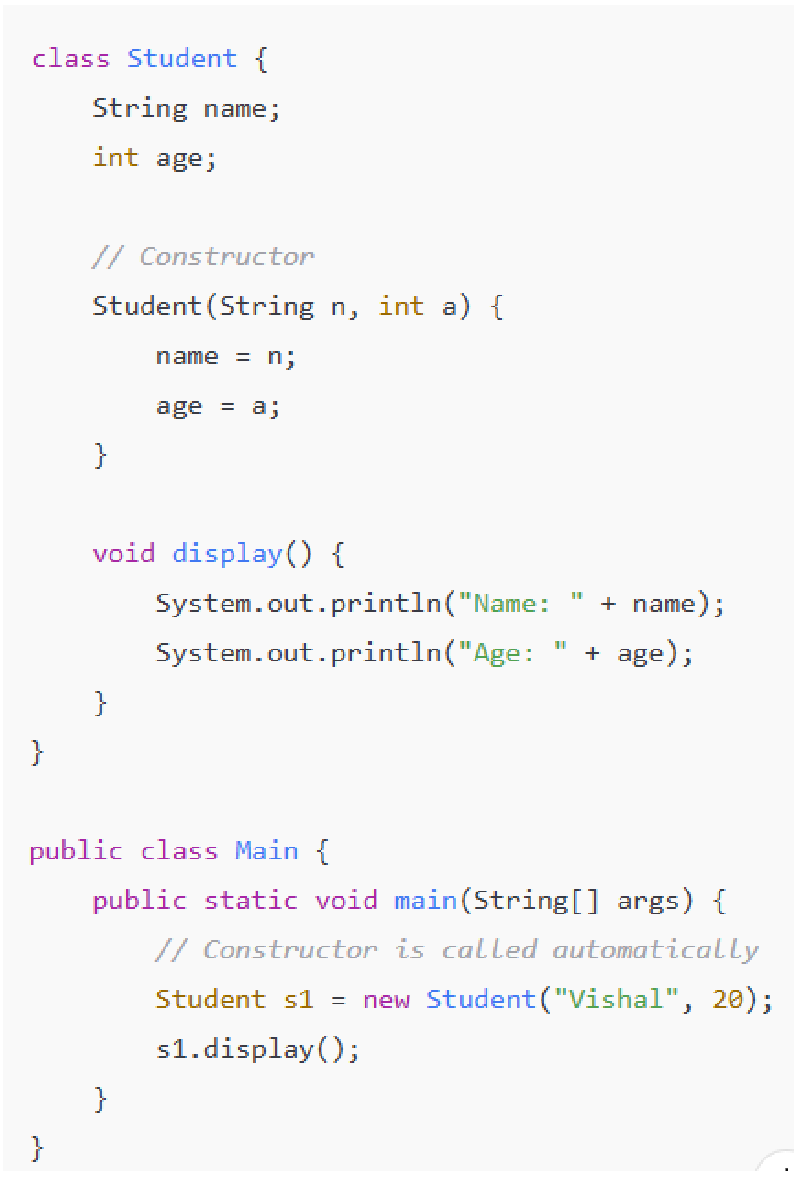
No return type:- Constructor do not have any return type, not even void. The main purpose of a constructor is to initialize the object not to return a value.
Automatically called an object creation:- When an object of a class is created the constructor is called automatically to initialize the object attributes.
Used to set Initial values for object Attributes:- Constructor are primarily used to set the initial state or values of an objects attributes when it is created.
Why do we need constructor in java:-
Constructor play a very important role, it ensures that an object is properly initialize before use.
Without Constructor:-

- Objects might have undefined or default values.
- Extra initialization method would be required.
- Risk of improper object state.

$\Rightarrow$ So constructor are used to assign values to the class variables at the time object creation, either explicitly done by the programmer or java itself.

When Java Constructor is called:-

- Each time an object is created using a new keyword, at least one constructor(it could be the default constructor) is invoked to assign initial values to the data members of the same class.
- The constructor of a class must have the same name as the class name in which it resides.
- A constructor in java cannot be abstract, final, static or synchronized.
- Access modifies can be used in constructor declaration to control its access which other class can call the constructor.



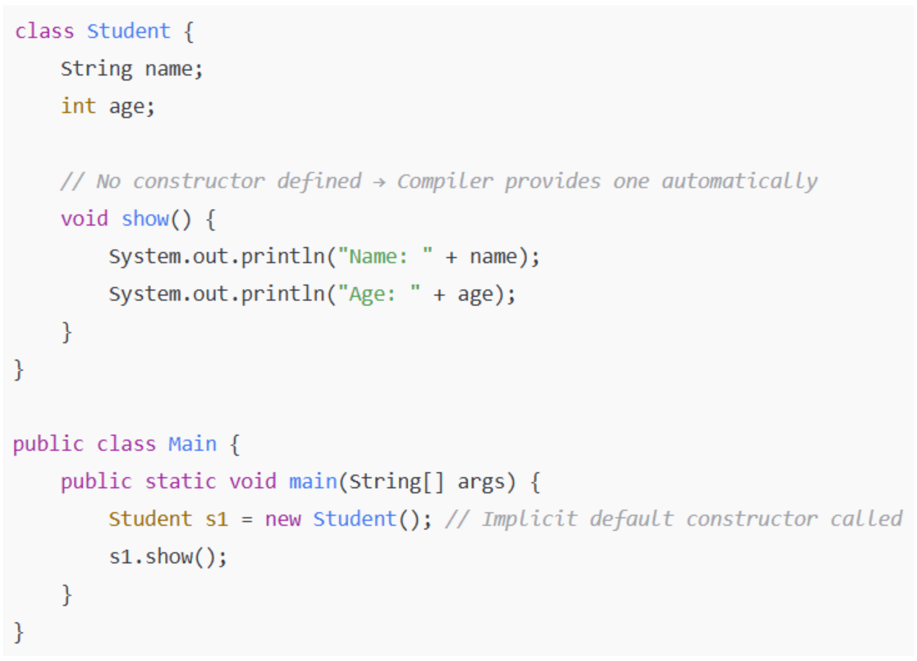
```java
class Student {
    String name;
    int age;

    // Constructor
    Student(String n, int a) {
        name = n;
        age = a;
    }

    void display() {
        System.out.println("Name:  " + name);
        System.out.println("Age:  " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        // Constructor is called automatically
        Student s1 = new Student("Vishal", 20);
        s1.display();
    }
}
```

Types of Constructors in Java

1). **Default constructor:-** A constructor that has no parameters is known as default constructor. A default constructor is invisible. And if we write a constructor with no arguments the compiler does not create a default constructor. Once you define a constructor (with or without parameter), the compiler no longer provides the default constructor. Defining a parameterized constructor does not automatically create a no-arguments, constructor, we must explicitly define if needed.

- The default constructor can be explicit or implicit.

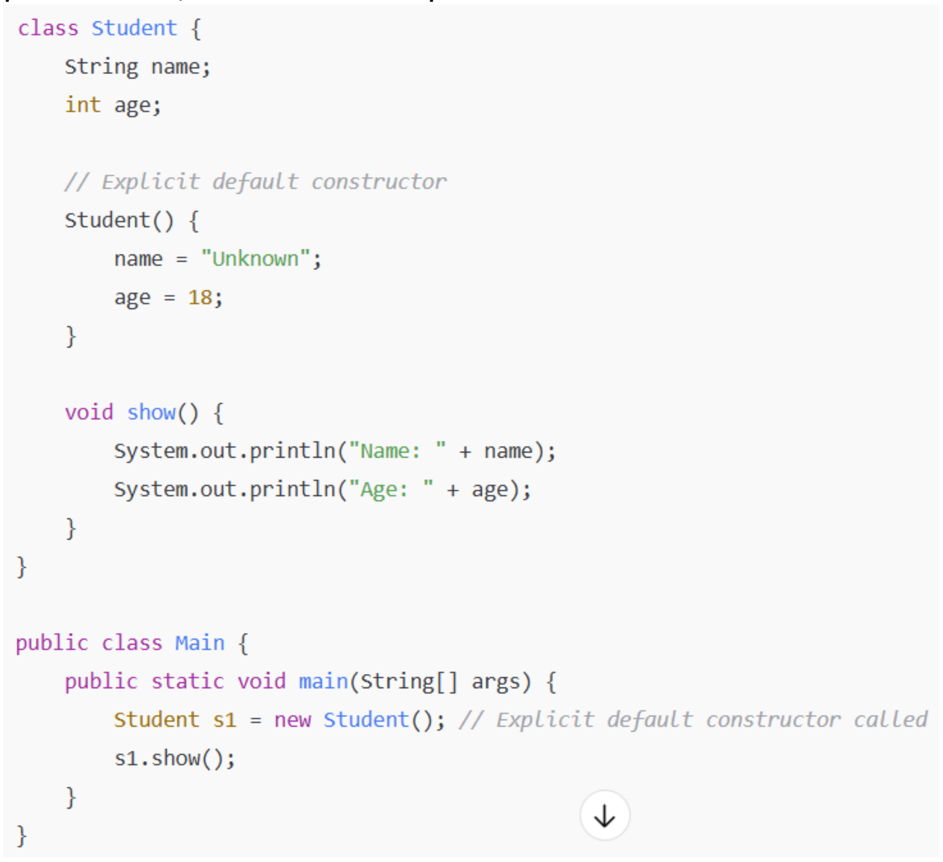
Implicit Default Constructor:- If no constructor is defined in a class, the java compiler automatically provides a default constructor. This constructor take the parameters and initializes the object with the default values, such as 0(numbers), null for objects.



```java
class Student {
    String name;
    int age;

    // No constructor defined → Compiler provides one automatically
    void show() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student(); // Implicit default constructor called
        s1.show();
    }
}
```

```
Name: null
Age: 0
```

<u>**Explicit Default constructor:-**</u> If we defines a constructor that takes no parameters, it's called an explicit default constructor.



```java
class Student {
    String name;
    int age;

    // Explicit default constructor
    Student() {
        name = "Unknown";
        age = 18;
    }

    void show() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student(); // Explicit default constructor called
        s1.show();
    }
}
```

```
Name: Unknown
Age: 18
```

2).Parameterized constructor:-

- A constructor that has parameters is known as parameterized constructor.
- Initialize object with specific values instead of default values.
- Avoid the need for setter method after object creation.
- Improve code readability by directly assigning values through constructor.

```
class Student {
    String name;
    int age;

    // Parameterized constructor
    Student(String n, int a) {
        name = n;
        age = a;
    }

    void show() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Vishal", 21);
        Student s2 = new Student("Rohit", 22);

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

```
Name: Vishal
Age: 21
Name: Rohit
Age: 22
```

Constructor Overloading :-

Like methods, constructors can also be overloaded.
Constructor Overloading in java means having more than one constructor in the same class with different parameters lists (type, number, or order of parameters).

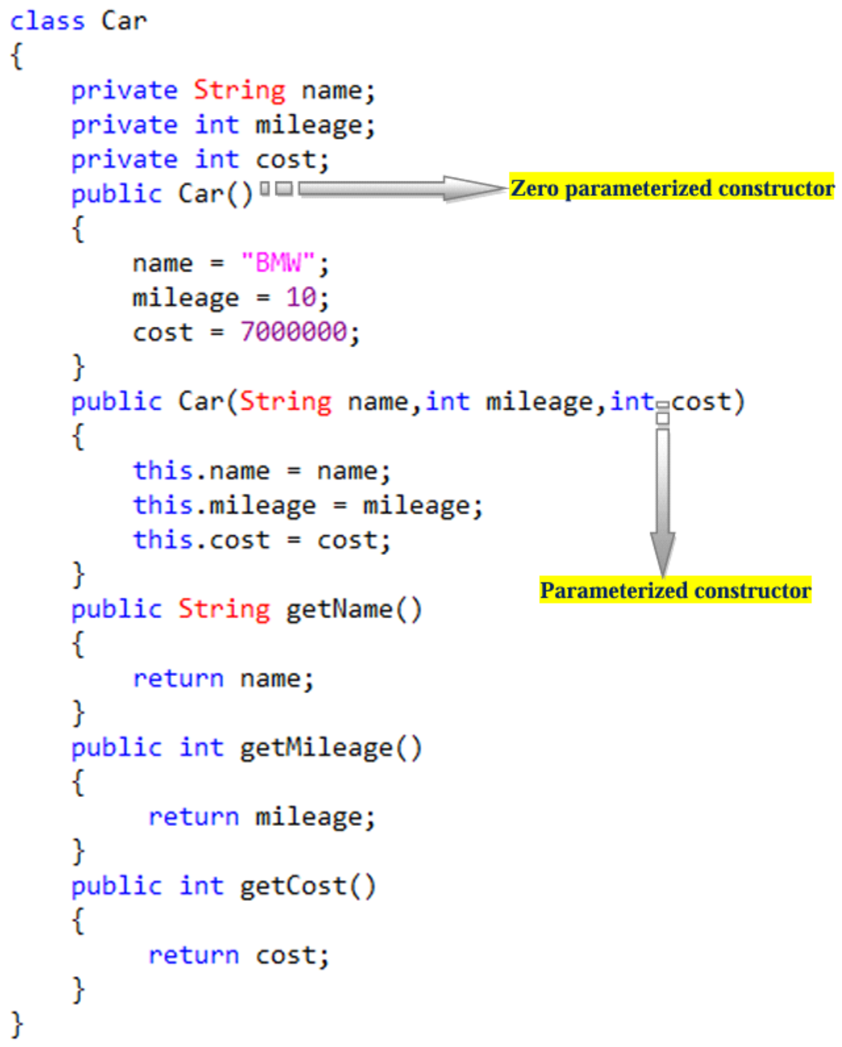
- It allow we to create object in multiple ways depending on what data is available.

Why use:-

- To provides flexibility during object creation.
- To support default, partial or full initialization of object data.
- A constructor in java cannot be abstract, final, static or synchronized.
- A access modifiers can be used in constructor for declaration to control its access which other class can call the constructor.
- Constructor overloading is required so that different objects can be initialized differently.

Rules :-

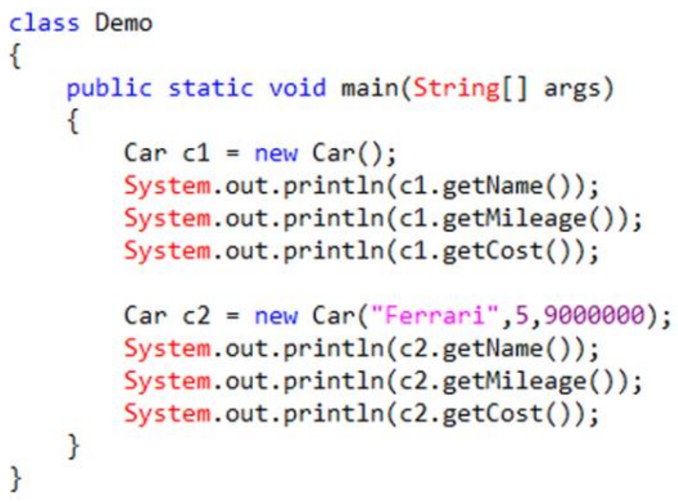
- Constructor must have different parameters lists.
- Java decides which constructor to call based on the number and type of arguments.



```
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
```

Constructor chaining in java:-

Constructor chaining is the process of calling one constructor from another constructor in the same class or superclass using the this() or super().

Two types of constructor chaining

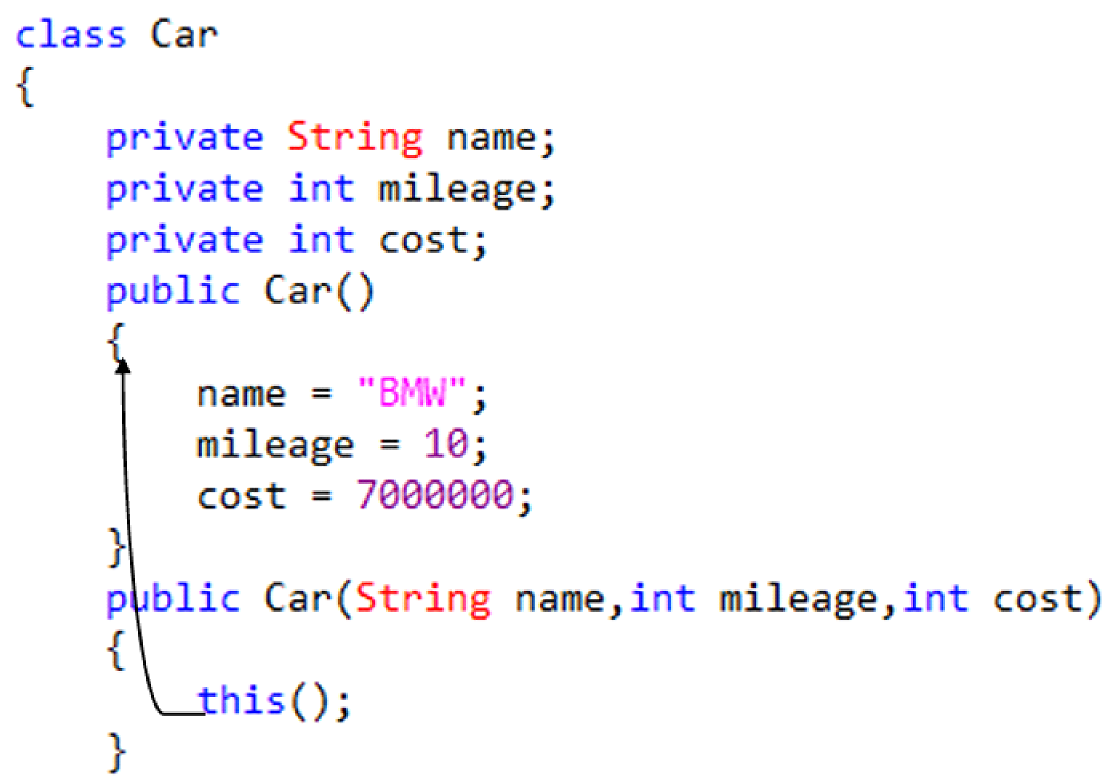
- Within the same class:- this() Local chaining
- B/W parent and child class:- super().

Local chaining

Local Chaining is the process of a constructor of a class calling another constructor of the same class.
Achieved using this keyword.
This() should compulsorily be the first line in the constructor.
It is used to refer to the current class method.
It is used to pass the current class instances as a parameter to the method.
It is used to pass the current class instances as a parameter to the constructor.
It is used to return the current class instances from the method.
It is used to invoke a constructor from another overloaded constructor in 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.



```
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());
     }
 }
```

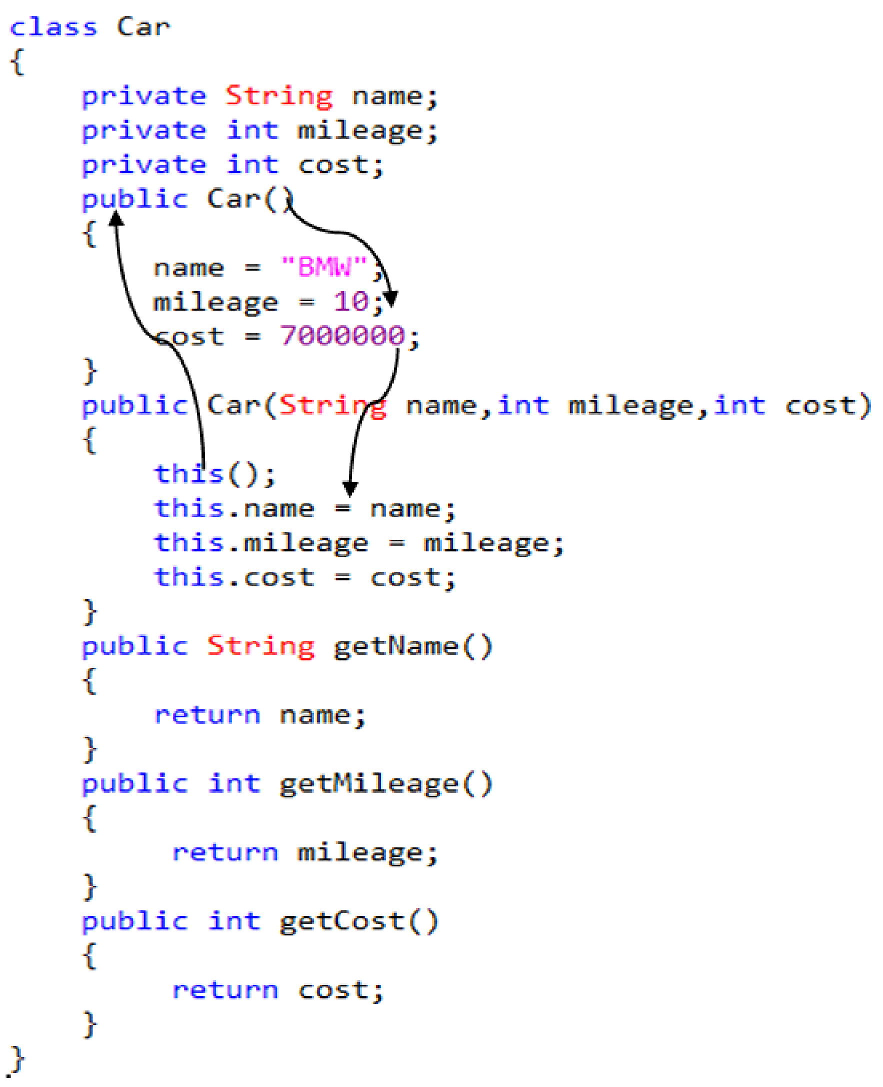
In the above example,

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

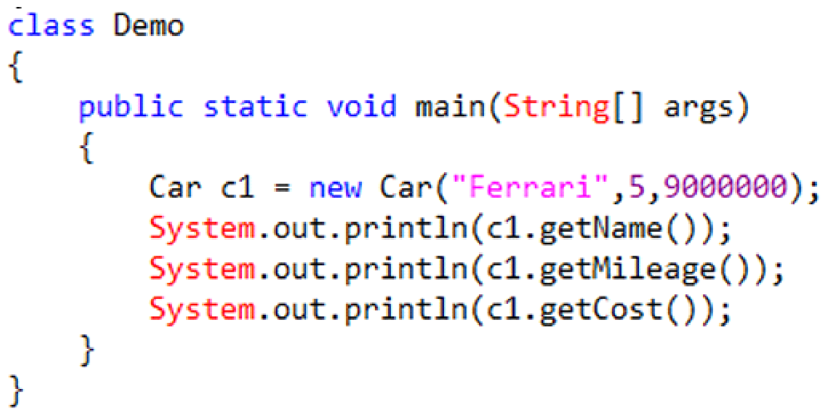
Replaced by this()
In parametrized constructor

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.

Example - 1

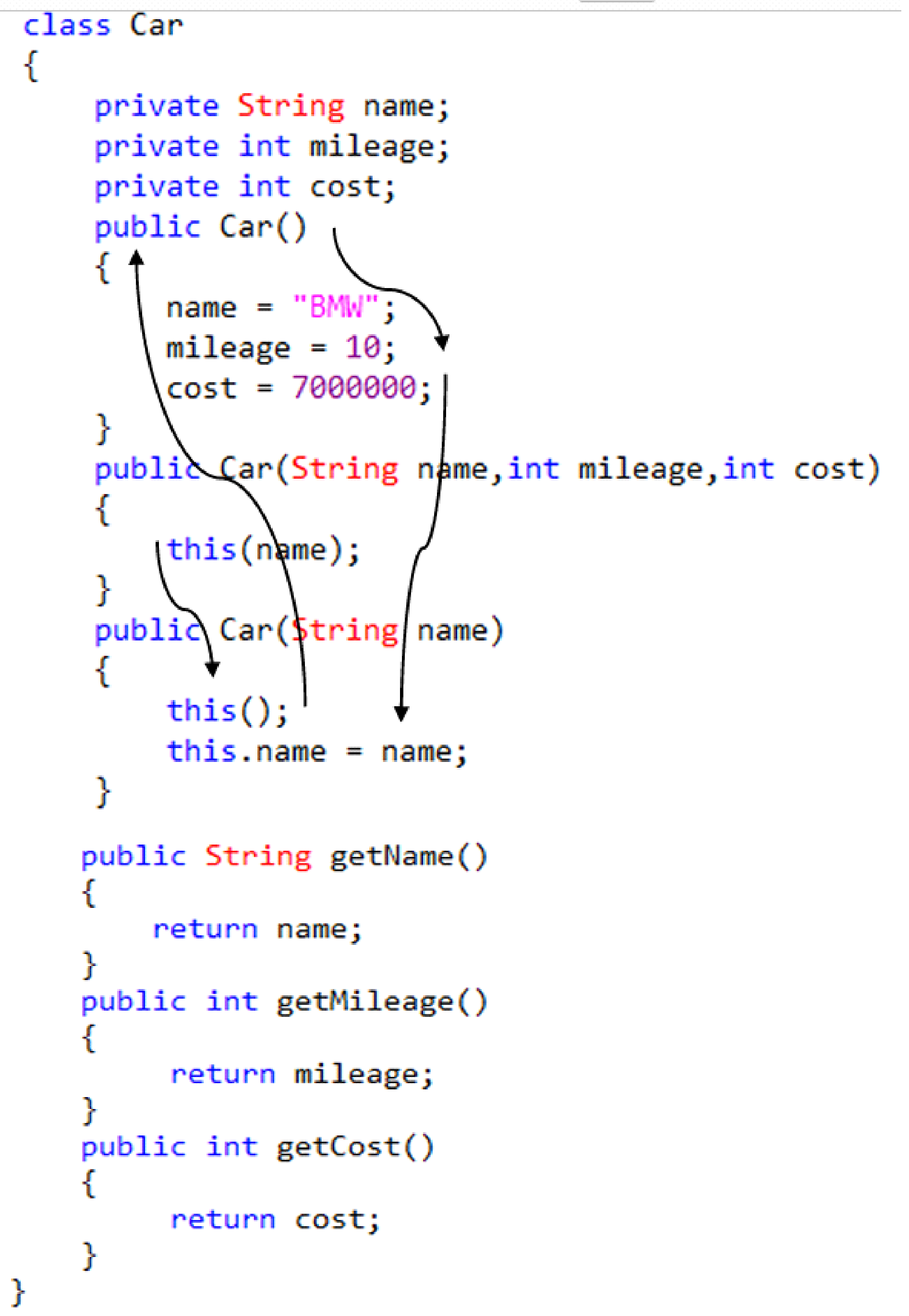


```
class Car
{
    private String name;
    private int mileage;
    private int cost;
    public Car()
    {
        name = "BMW";
        mileage = 10;
        cost = 7000000;
    }
    public Car(String name,int mileage,int cost)
    {
        this();
        this.name = name;
        this.mileage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



```
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;
    }
}
```

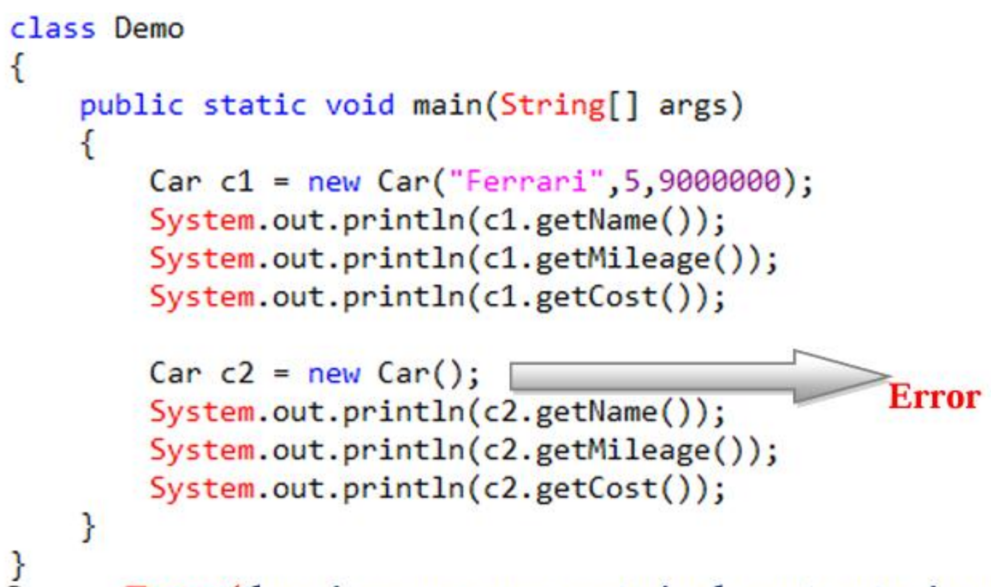
```
class Demo
{
    public static void main(String[] args)
    {
        Car c1 = new Car("Ferrari",5,9000000);
        System.out.println(c1.getName());
        System.out.println(c1.getMileage());
        System.out.println(c1.getCost());
    }
}
```

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().

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)**

Sl. No.	Constructor	Method
1	Constructor name should be same as class name.	Method name may or may not be same as class name.
2	Constructor never returns any value so it has no return type.	Method must have a return type in Java and returns only a single value.
3	There are 2 types of constructor available in Java.	Java supports 6 types of method.
4	Constructor only can be called by new keyword in Java	Method can be called by class name, object name or directly.
5	If there is no constructor designed by user then the Java compiler automatically provides the default constructor.	Javac never provides any method by default.
6	Constructor cannot be overridden in Java.	Method can be overridden.
7	Constructor cannot be declared as static.	Method can be declared as Static.