CodInClub

Powered By BridgeLabz

What is a Constructor?

A **constructor** in Java is a special method that initializes an object when it is created. Unlike regular methods, constructors:

1. Have the same name as the class.
2. Do not have a return type.
3. Are automatically called when the object is instantiated using the `new` keyword.

Key Features of Constructors

1. **Automatic Invocation**: Called during object creation.
2. **Overloading Support**: You can define multiple constructors with different parameter lists.
3. **Initialization**: Primarily used to initialize object attributes.

Coffee Shop Order System

Let's create a **CoffeeShopOrder** class to manage customer orders in a coffee shop.

Sample Program: CoffeeShopOrder Class

```
class CoffeeShopOrder {
    // Attributes
    String customerName;
    String coffeeType;
    int quantity;
    double totalPrice;

    // Default Constructor
    CoffeeShopOrder() {
        customerName = "Guest";
        coffeeType = "Regular";
        quantity = 1;
        totalPrice = calculatePrice();
```

```
    }

    // Parameterized Constructor
    CoffeeShopOrder(String customerName, String coffeeType, int
quantity) {
        this.customerName = customerName;
        this.coffeeType = coffeeType;
        this.quantity = quantity;
        this.totalPrice = calculatePrice();
    }

    // Copy Constructor
    CoffeeShopOrder(CoffeeShopOrder previousOrder) {
        this.customerName = previousOrder.customerName;
        this.coffeeType = previousOrder.coffeeType;
        this.quantity = previousOrder.quantity;
        this.totalPrice = previousOrder.totalPrice;
    }

    // Method to calculate price
    private double calculatePrice() {
        double pricePerCup = switch (coffeeType.toLowerCase()) {
            case "latte" -> 5.0;
            case "espresso" -> 4.0;
            case "cappuccino" -> 4.5;
            default -> 3.0; // Regular coffee
        };
        return pricePerCup * quantity;
    }

    // Display Order Details
    void displayOrderDetails() {
        System.out.println("Customer Name: " + customerName);
        System.out.println("Coffee Type: " + coffeeType);
        System.out.println("Quantity: " + quantity);
        System.out.println("Total Price: $" + totalPrice);
    }
```

```
}

public class Main {
    public static void main(String[] args) {
        // Default Constructor
        CoffeeShopOrder order1 = new CoffeeShopOrder();
        System.out.println("Order 1:");
        order1.displayOrderDetails();

        // Parameterized Constructor
        CoffeeShopOrder order2 = new CoffeeShopOrder("Alice", "Latte",
3);
        System.out.println("\nOrder 2:");
        order2.displayOrderDetails();

        // Copy Constructor
        CoffeeShopOrder order3 = new CoffeeShopOrder(order2);
        System.out.println("\nOrder 3 (Copy of Order 2):");
        order3.displayOrderDetails();
    }
}
```

Output

```
Order 1:
Customer Name: Guest
Coffee Type: Regular
Quantity: 1
Total Price: $3.0

Order 2:
Customer Name: Alice
Coffee Type: Latte
Quantity: 3
Total Price: $15.0
```

```
Order 3 (Copy of Order 2):
Customer Name: Alice
Coffee Type: Latte
Quantity: 3
Total Price: $15.0
```

Explanation of the Program

1. **Default Constructor**:
   - Initializes attributes with default values.
   - In this example, if no parameters are provided, the customer is treated as a "Guest" ordering one "Regular" coffee.
2. **Parameterized Constructor**:
   - Accepts values for `customerName`, `coffeeType`, and `quantity`.
   - Dynamically calculates the total price based on the coffee type and quantity.
3. **Copy Constructor**:
   - Creates a new object by copying the attributes of an existing object.
   - Used here to duplicate an order for simplicity.
4. **Switch Case for Coffee Prices**:
   - Dynamically assigns the price per cup based on the coffee type.

Constructor Overloading in Action

The program demonstrates constructor overloading by using multiple constructors:

1. The **default constructor** initializes default values.
2. The **parameterized constructor** customizes the initialization.
3. The **copy constructor** replicates an existing object's attributes.

CodInClub
Powered By BridgeLabz

Access Modifiers in Java

What are Access Modifiers?
Access modifiers in Java control the visibility and accessibility of classes, methods, and attributes.

Types of Access Modifiers:

Modifier	Class	Same Package	Subclass	Outside Package
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	No
default	Yes	Yes	No	No
private	Yes	No	No	No

Examples:

```
class Example {
    public int publicVariable = 10;       // Accessible everywhere
    protected int protectedVariable = 20; // Accessible in the same
package and subclasses
    int defaultVariable = 30;             // Accessible in the same
package
    private int privateVariable = 40;     // Accessible only within
the class

    void displayAccess() {
        System.out.println("Public: " + publicVariable);
        System.out.println("Protected: " + protectedVariable);
```

```
        System.out.println("Default: " + defaultVariable);
        System.out.println("Private: " + privateVariable);
    }
}
```

Instance vs. Class Variables and Methods

Instance Variables and Methods

- **Instance Variables**:
  Belong to an object; unique to each instance.
  Example: `String name;`

Instance Methods:
Operate on instance variables and can be accessed only via objects.
Example:

```
void displayDetails() {
    System.out.println(name);
}
```

Class Variables and Methods

Class Variables:
Declared using the `static` keyword. Shared among all objects.
Example:

```
static int totalStudents;
```

Class Methods:
Operate on static variables and can be accessed without creating an object.
Example:

```
static void incrementStudents() {
    totalStudents++;
}
```

Comparison Table:

Aspect	Instance Variables/Methods	Class Variables/Methods
Belongs To	A specific object	The class itself
Access	Through object	Through the class name or object
Keyword	No special keyword	Declared using `static`
Example	`student.name`	`Student.totalStudents`

Sample Program:

```
class School {
    String studentName;        // Instance Variable
    static int totalStudents;  // Class Variable

    // Constructor
    School(String studentName) {
        this.studentName = studentName;
        totalStudents++;       // Increment total students
    }

    // Instance Method
    void displayStudent() {
        System.out.println("Student Name: " + studentName);
    }

    // Class Method
    static void displayTotalStudents() {
        System.out.println("Total Students: " + totalStudents);
    }
}

public class Main {
    public static void main(String[] args) {
```

```
        School student1 = new School("Alice");
        School student2 = new School("Bob");

        student1.displayStudent();
        student2.displayStudent();

        School.displayTotalStudents();  // Accessing class method
    }
}
```

Output:

```
Student Name: Alice
Student Name: Bob
Total Students: 2
```