CodInClub

Powered By BridgeLabz

‘Best Programming Practice

1. Use meaningful class names (e.g., Student, Employee) and method names (e.g., displayDetails).
2. Encapsulate data using private fields and provide getter and setter methods.
3. Follow proper naming conventions (camelCase for attributes and methods).
4. Always provide constructors to initialize class attributes.
5. Use comments for clarity and better readability.

Sample Program 1: Food Delivery App

Real-World Analogy

Imagine a **food delivery app** like Swiggy or Uber Eats. The app deals with **restaurants**, and each restaurant has specific details like its name, location, and the food items it serves.

Step 1: Define the Class

The Restaurant class represents the blueprint for creating restaurant objects.

```
// Class Definition
public class Restaurant {
    // Fields (Attributes)
    private String name;
    private String location;
    private String[] foodItems;

    // Constructor
    public Restaurant(String name, String location, String[] foodItems) {
        this.name = name;
        this.location = location;
        this.foodItems = foodItems;
    }

    // Method to display restaurant details
    public void displayDetails() {
        System.out.println("Restaurant Name: " + name);
        System.out.println("Location: " + location);
        System.out.println("Food Items: ");
        for (String item : foodItems) {
            System.out.println("- " + item);
```

```
        }
    }

    // Method to check if a food item is available
    public boolean isFoodAvailable(String food) {
        for (String item : foodItems) {
            if (item.equalsIgnoreCase(food)) {
                return true;
            }
        }
        return false;
    }
}
```

Step 2: Create Objects from the Class

Use the class to create specific restaurant objects.

```
// Main Class to Test
public class Main {
    public static void main(String[] args) {
        // Define food items for restaurants
        String[] foodItems1 = {"Pizza", "Pasta", "Burger"};
        String[] foodItems2 = {"Sushi", "Ramen", "Tempura"};

        // Create Restaurant objects
        Restaurant restaurant1 = new Restaurant("Italian Delight", "Downtown", foodItems1);
        Restaurant restaurant2 = new Restaurant("Tokyo Treats", "Uptown", foodItems2);

        // Display details of each restaurant
        System.out.println("=== Restaurant 1 ===");
        restaurant1.displayDetails();
        System.out.println("\n=== Restaurant 2 ===");
        restaurant2.displayDetails();

        // Check food availability
        System.out.println("\nChecking Food Availability:");
                        System.out.println("Is Pasta available in Italian Delight? " +
restaurant1.isFoodAvailable("Pasta"));
                        System.out.println("Is Sushi available in Italian Delight? " +
restaurant1.isFoodAvailable("Sushi"));
```

```
    }
}
```

Step 3: Output

```
=== Restaurant 1 ===
Restaurant Name: Italian Delight
Location: Downtown
Food Items:
- Pizza
- Pasta
- Burger

=== Restaurant 2 ===
Restaurant Name: Tokyo Treats
Location: Uptown
Food Items:
- Sushi
- Ramen
- Tempura

Checking Food Availability:
Is Pasta available in Italian Delight? true
Is Sushi available in Italian Delight? false
```

In-depth explanation of Key Aspects

1. Fields (Attributes)

- Fields store the data for the class.
- Example: `name`, `location`, and `foodItems` represent the state of a restaurant.

2. Constructor

- A constructor initializes the fields when an object is created.
- Example: The `Restaurant` constructor sets `name`, `location`, and `foodItems`.

3. Methods

- Methods define the behavior of the objects.
- Example:
    - `displayDetails()`: Displays the details of a restaurant.
    - `isFoodAvailable(String food)`: Checks if a specific food item is available.

4. Encapsulation

- The fields are marked as private and accessed using methods to ensure controlled data access and modification.

5. Object Creation

- Objects are created using the new keyword.

Example:
Restaurant restaurant1 = new Restaurant("Italian Delight", "Downtown", foodItems1);

6. Memory Allocation

- Each object has its own memory space for attributes but shares methods.

CodInClub

Powered By BridgeLabz

Level 1 Practice Programs

1. **Program to Display Employee Details**

   Problem Statement: Write a program to create an Employee class with attributes name, id, and salary. Add a method to display the details.

2. **Program to Compute Area of a Circle**

   Problem Statement: Write a program to create a Circle class with an attribute radius. Add methods to calculate and display the area and circumference of the circle.

3. **Program to Track Inventory of Items**

   Problem Statement: Create an Item class with attributes itemCode, itemName, and price. Add a method to display item details and calculate the total cost for a given quantity.

4. **Program to Handle Mobile Phone Details**

   Problem Statement: Create a MobilePhone class with attributes brand, model, and price. Add a **method** to display all the details of the phone. The MobilePhone class uses attributes to store the phone's characteristics. The **method** is used to retrieve and display this information for each **object**.

Level 2 Practice Programs

1. **Program to Simulate Student Report**

   Problem Statement: Create a Student class with attributes name, rollNumber, and marks. Add two methods:

   - To calculate the grade based on the marks.
   - To display the student's details and grade.

   Explanation: The Student class organizes all relevant details about a student as attributes. Methods are used to calculate the grade and provide a way to display all information.

2. **Program to Simulate an ATM**

Problem Statement: Create a BankAccount class with attributes accountHolder, accountNumber, and balance. Add methods for:

- Depositing money.
- Withdrawing money (only if sufficient balance exists).
- Displaying the current balance.

Explanation: The BankAccount class stores bank account details as attributes. The methods allow interaction with these attributes to modify and view the account's state.

3. **Program to Check Palindrome String**

Problem Statement: Create a PalindromeChecker class with an attribute text. Add **methods** to:

- Check if the text is a palindrome.
- Display the result.

Explanation: The PalindromeChecker class holds the text attribute. The **methods** operate on this attribute to verify its palindrome status and display the result.

4. **Program to Model a Movie Ticket Booking System**

Problem Statement: Create a MovieTicket class with attributes movieName, seatNumber, and price. Add **methods** to:

- Book a ticket (assign seat and update price).
- Display ticket details.

Explanation: The MovieTicket class organizes ticket information with attributes. The **methods** handle booking logic and display ticket details.

5. **Program to Simulate a Shopping Cart**

Problem Statement: Create a CartItem class with attributes itemName, price, and quantity. Add **methods** to:

- Add an item to the cart.
- Remove an item from the cart.
- Display the total cost.

Explanation: The CartItem class models a shopping cart item. The **methods** handle cart operations like adding or removing items and calculating the total cost.


CodInClub
Powered By
BridgeLabz