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.

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.

```java
class Employee {
    String name;
    int id;
    double salary;

    Employee(String name, int id, double salary) {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }

    void displayDetails() {
        System.out.println("Employee ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Salary: $" + salary);
    }

    public static void main(String[] args) {
        Employee emp = new Employee("John Doe", 101, 50000);
        emp.displayDetails();
    }
}
```

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.

```java
import java.util.Scanner;

class Circle {
    double radius;

    Circle(double radius) {
        this.radius = radius;
    }

    double calculateArea() {
        return Math.PI * radius * radius;
    }

    double calculateCircumference() {
        return 2 * Math.PI * radius;
    }

    void displayDetails() {
        System.out.println("Radius: " + radius);
        System.out.println("Area: " + calculateArea());
        System.out.println("Circumference: " + calculateCircumference());
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter radius: ");
        double radius = sc.nextDouble();
        Circle circle = new Circle(radius);
        circle.displayDetails();
        sc.close();
    }
}
```

3. **Program to Handle Book Details**
   Problem Statement: Write a program to create a `Book` class with attributes `title`, `author`, and `price`. Add a method to display the book details.

```
class Book {
    String title, author;
    double price;

    Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    void displayDetails() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: $" + price);
    }

    public static void main(String[] args) {
        Book book = new Book("Java Programming", "James Gosling", 29.99);
        book.displayDetails();
    }
}
```

4. **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.

```
class Item {
    int itemCode;
    String itemName;
    double price;

    Item(int itemCode, String itemName, double price) {
        this.itemCode = itemCode;
        this.itemName = itemName;
        this.price = price;
    }

    double calculateTotalCost(int quantity) {
        return price * quantity;
    }

    void displayDetails(int quantity) {
        System.out.println("Item Code: " + itemCode);
        System.out.println("Item Name: " + itemName);
        System.out.println("Price per unit: $" + price);
        System.out.println("Total Cost for " + quantity + " units: $" +
calculateTotalCost(quantity));
    }

    public static void main(String[] args) {
        Item item = new Item(101, "Laptop", 750);
        item.displayDetails(2);
    }
}
```

5. **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**.

```
class MobilePhone {
    String brand, model;
    double price;

    MobilePhone(String brand, String model, double price) {
        this.brand = brand;
        this.model = model;
        this.price = price;
    }
    void displayDetails() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Price: $" + price);
    }

    public static void main(String[] args) {
        MobilePhone phone = new MobilePhone("Samsung", "Galaxy S21",
999.99);
        phone.displayDetails();
    }
}
```

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.

```java
class Student {
    String name;
    int rollNumber;
    double marks;

    Student(String name, int rollNumber, double marks) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.marks = marks;
    }

    String calculateGrade() {
        if (marks >= 90) return "A";
        else if (marks >= 80) return "B";
        else if (marks >= 70) return "C";
        else if (marks >= 60) return "D";
        else return "F";
    }

    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Marks: " + marks);
        System.out.println("Grade: " + calculateGrade());
    }

    public static void main(String[] args) {
        Student student = new Student("Alice", 123, 85);
        student.displayDetails();
    }
}
```

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.

```
class BankAccount {
    String accountHolder;
    int accountNumber;
    double balance;

    BankAccount(String accountHolder, int accountNumber, double balance) {
        this.accountHolder = accountHolder;
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: $" + amount + ". New Balance: $" +
balance);
    }

    void withdraw(double amount) {
        if (amount > balance) {
            System.out.println("Insufficient balance!");
        } else {
            balance -= amount;
            System.out.println("Withdrawn: $" + amount + ". New Balance: $"
+ balance);
        }
    }

    void displayBalance() {
        System.out.println("Current Balance: $" + balance);
    }

    public static void main(String[] args) {
        BankAccount account = new BankAccount("Bob", 456789, 1000);
        account.deposit(500);
        account.withdraw(200);
        account.displayBalance();
    }
}
```

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.

```
import java.util.Scanner;

class PalindromeChecker {
    String text;

    PalindromeChecker(String text) {
        this.text = text;
    }

    boolean isPalindrome() {
        int left = 0, right = text.length() - 1;
        while (left < right) {
            if (text.charAt(left) != text.charAt(right)) return false;
            left++;
            right--;
        }
        return true;
    }

    void displayResult() {
        if (isPalindrome()) {
            System.out.println(text + " is a palindrome.");
        } else {
            System.out.println(text + " is not a palindrome.");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter text: ");
        String text = sc.nextLine();
        PalindromeChecker checker = new PalindromeChecker(text);
        checker.displayResult();
        sc.close();
    }
}
```

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.

```java
class MovieTicket {
    String movieName;
    int seatNumber;
    double price;

    MovieTicket(String movieName, int seatNumber, double price) {
        this.movieName = movieName;
        this.seatNumber = seatNumber;
        this.price = price;
    }

    void displayTicketDetails() {
        System.out.println("Movie: " + movieName);
        System.out.println("Seat Number: " + seatNumber);
        System.out.println("Price: $" + price);
    }

    public static void main(String[] args) {
        MovieTicket ticket = new MovieTicket("Avengers", 12, 15.50);
        ticket.displayTicketDetails();
    }
}
```

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.

```java
import java.util.Scanner;
```

```java
class CartItem {
    String itemName;
    double price;
    int quantity;


    public CartItem(String itemName, double price, int quantity) {
        this.itemName = itemName;
        this.price = price;
        this.quantity = quantity;
    }


    public double getTotalCost() {
        return price * quantity;
    }


    public void displayItem() {
        System.out.println("Item Name: " + itemName);
        System.out.println("Price: $" + price);
        System.out.println("Quantity: " + quantity);
        System.out.println("Total Cost: $" + getTotalCost());
    }
}

public class ShoppingCart {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        System.out.print("Enter item name: ");
        String name = scanner.nextLine();

        System.out.print("Enter item price: ");
        double price = scanner.nextDouble();

        System.out.print("Enter quantity: ");
        int quantity = scanner.nextInt();

        CartItem item = new CartItem(name, price, quantity);
        System.out.println("\nItem added to the cart:");
        item.displayItem();

        scanner.close();
    }
}
```