| | | |
|---|---|---|
| **Course Code** | **:** | **MCS-024** |
| **Course Title** | **:** | **Object Oriented Technologies and Java Programming** |
| **Assignment Number** | **:** | **BCA (IV)/024/Assignment/2024-25** |
| **Maximum Marks** | **:** | **100%** |
| **Last Date of Submission** | **:** | **31st October, 2024 (For July session)**<br>**30th April, 2025 (For January session)** |

**There are Three questions in this assignment which carry 80 marks, in total. Rest 20 marks are for viva-voce. Answer all the questions. Give appropriate comments in programs to increase understandability. Wherever required, you may write java program, run it on machine and take its output as part of solution. Please go through the guidelines regarding assignments given in the Programme Guide for the format of presentation.**

**Q1:** Give an example of a Project that is suitable for application of Procedural Programming rather than Object Oriented Programming. Justify your answer point wise in detail. **(30 marks)**

**Q2:** Give an example of a Project that is suitable for application of Object Oriented Programming rather than Procedural Programming. Justify your answer point wise in detail. **(30 marks)**

**Q3:** Give an example of a Project that is suitable for application of Object Oriented Programming and usage of Java Programming Language rather than any other Object Oriented Programming Language. Justify your answer point wise in detail. **(20 marks)**

# Q1: Procedural Programming vs Object Oriented Programming

Procedural programming is a programming paradigm that emphasizes the use of procedures or functions to solve problems. In contrast, object-oriented programming (OOP) is a programming paradigm that focuses on creating objects that contain both data and methods. To determine whether a project is more suitable for procedural programming or object-oriented programming, we need to consider the following factors:

## 1. Data and functionality

In procedural programming, data and functionality are separate, and data is passed as arguments to functions. In OOP, data and functionality are combined into objects, and objects interact with each other through methods.

## 2. Modularity

Procedural programming emphasizes breaking down a problem into smaller, reusable modules or functions. OOP focuses on creating reusable objects that encapsulate data and methods.

## 3. Inheritance

OOP supports inheritance, which allows objects to inherit properties and methods from parent objects. Procedural programming does not have a built-in mechanism for inheritance.

## 4. Abstraction

OOP supports abstraction, which allows you to hide implementation details and focus on the essential features of an object. Procedural programming does not have a built-in mechanism for abstraction.

# 5. Polymorphism

OOP supports polymorphism, which allows objects to take on multiple forms. Procedural programming does not have a built-in mechanism for polymorphism.An example of a project that is suitable for procedural programming is a simple calculator program. In this case, the program would consist of a series of functions that perform basic arithmetic operations (addition, subtraction, multiplication, and division). The data (the numbers to be operated on) would be passed as arguments to these functions, and the results would be returned. This approach is simple and straightforward, and it does not require the use of objects or OOP concepts.On the other hand, a project that is more suitable for OOP is a complex system with multiple interacting components, such as a banking system or a game. In these types of projects, objects can be used to represent different entities (such as accounts, transactions, or players) and their interactions. Objects can also be used to encapsulate data and methods, making the code more modular and easier to maintain.In summary, procedural programming is suitable for simple programs with a clear flow of execution and minimal data requirements, while OOP is more suitable for complex systems with multiple interacting components and data requirements.

Here's an example of a simple calculator program in Java using procedural programming

```java
import java.util.Scanner;

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

        System.out.print("Enter the first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter the second number: ");
        double num2 = scanner.nextDouble();

        System.out.print("Enter the operation (+, -, *, /): ");
        char operator = scanner.next().charAt(0);

        double result = 0;

        switch (operator) {
            case '+':
                result = add(num1, num2);
                break;
            case '-':
                result = subtract(num1, num2);
                break;
            case '*':
                result = multiply(num1, num2);
                break;
            case '/':
                result = divide(num1, num2);
                break;
```

```
            default:
                System.out.println("Invalid operator!");
                return;
        }

        System.out.println("Result: " + result);
    }

    public static double add(double a, double b) {
        return a + b;
    }

    public static double subtract(double a, double b) {
        return a - b;
    }

    public static double multiply(double a, double b) {
        return a * b;
    }

    public static double divide(double a, double b) {
        if (b == 0) {
            System.out.println("Error: Division by zero!");
            return 0;
        }
        return a / b;
    }
}
```

## Q2: Give an example of a Project that is suitable for application of Object Oriented Programming rather than Procedural Programming. Justify your answer point wise in detail.

A suitable project for the application of Object Oriented Programming (OOP) is a **Library Management System**. This project involves managing various aspects of a library, including books, members, and transactions. Below are the justifications for using OOP in this context:

### 1. Real-World Modeling

OOP allows for the modeling of real-world entities. In a library management system, entities such as `Book`, `Member`, and `Transaction` can be represented as objects. Each object can encapsulate relevant attributes (data) and behaviors (methods), making the system intuitive and aligned with real-world interactions.

## 2. Encapsulation

Encapsulation is a core principle of OOP that allows for data hiding and protection. In the library system, sensitive data such as member information and book availability can be encapsulated within their respective classes. This prevents unauthorized access and modifications, enhancing security.

## 3. Inheritance

OOP supports inheritance, enabling the creation of a hierarchy of classes. For example, you could have a base class `User` and derived classes such as `Member` and `Librarian`. This allows for code reusability and the ability to extend functionality without modifying existing code, which is beneficial for maintaining and updating the system.

## 4. Polymorphism

Polymorphism allows for methods to be overridden in derived classes. In the library system, a method like `issueBook()` can behave differently for `Member` and `Librarian` classes. This flexibility simplifies the code and allows for more dynamic interactions within the system.

## 5. Modularity

OOP promotes modularity, making it easier to manage and maintain large codebases. Each class can be developed and tested independently, facilitating collaboration among multiple developers. In a library system, different modules can handle book management, member management, and transaction processing separately.

## 6. Ease of Maintenance and Scalability

As requirements evolve, OOP makes it easier to modify and extend the system. New features, such as adding a `Reservation` class for reserving books, can be integrated without affecting existing functionality. This adaptability is crucial for a library system that may need to evolve with user needs.

## Example Class Structure

Here's a simplified example of how classes might be structured in a Library Management System:

```java
class Book {
    private String title;
    private String author;
    private boolean isAvailable;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isAvailable = true;
    }

    public void issue() {
        isAvailable = false;
    }

    public void returnBook() {
        isAvailable = true;
    }

    public boolean checkAvailability() {
        return isAvailable;
    }
}

class Member {
    private String name;
    private int memberId;
```

```java
    public Member(String name, int memberId) {
        this.name = name;
        this.memberId = memberId;
    }

    public void borrowBook(Book book) {
        if (book.checkAvailability()) {
            book.issue();
            System.out.println(name + " borrowed " + book.title);
        } else {
            System.out.println(book.title + " is not available.");
        }
    }
}
```

In this example, the `Book` class encapsulates the properties and behaviors related to books, while the `Member` class encapsulates member-related functionalities. This structure exemplifies how OOP can effectively manage complex systems by organizing code around real-world concepts.

# Q3: **Give an example of a Project that is suitable for application of Object Oriented Programming and usage of Java Programming Language rather than any other Object Oriented Programming Language. Justify your answer point wise in detail.**

**Project Example:** A **distributed online auction system**.

**Justification:**

1. **Java's Platform Independence:**
    - Java's "write once, run anywhere" capability is crucial for a distributed auction system that needs to operate on various platforms (e.g., web servers, client machines). Java's platform independence ensures that the application runs consistently across different environments.
2. **Rich Standard Library and APIs:**
    - Java offers a robust set of libraries and APIs for building complex applications. For instance, the Java Standard Edition includes libraries for networking (Java Networking API), GUI development (JavaFX, Swing), and data management (Java Database Connectivity). These libraries are beneficial for developing a feature-rich auction system.
3. **Built-in Concurrency Support:**
    - Java’s built-in support for multithreading and concurrency is vital for handling multiple simultaneous bids, user interactions, and data processing in an auction system. Java's concurrency utilities, such as the `ExecutorService` and `synchronized` blocks, facilitate smooth operation under high load.
4. **Security Features:**

- Security is a critical aspect of an online auction system due to sensitive user information and transactions. Java provides comprehensive security features, including secure communication (SSL/TLS), encryption libraries, and robust authentication mechanisms.

5. **Object-Oriented Design Benefits:**
    - The auction system can be effectively modeled using object-oriented principles. For example:
        - **Classes and Objects:** Entities such as `User`, `Item`, `Bid`, and `Auction` can be modeled as classes with their attributes and methods.
        - **Inheritance:** Specialized classes can inherit from base classes. For instance, `PremiumUser` might extend `User` with additional privileges.
        - **Polymorphism:** Different types of auction items (e.g., physical goods, digital items) can be handled polymorphically through a common interface.
        - **Encapsulation:** Sensitive data, such as user bids and personal information, can be encapsulated within classes to ensure controlled access.
6. **Integration with Java Enterprise Technologies:**
    - For a fully-featured auction system, Java Enterprise Edition (Java EE) technologies can be used for web services, database integration, and enterprise-level scalability. Technologies such as Java Servlets, JSP, and JPA (Java Persistence API) support building scalable and maintainable applications.

**Sample Code (Java):**

Here's a simplified example of how an auction system might be modeled using Java:

```
import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.ConcurrentMap;


// Base class for Auction Items

abstract class AuctionItem {

   private String itemName;

   private double currentBid;


   public AuctionItem(String itemName) {

      this.itemName = itemName;

      this.currentBid = 0.0;

   }
```

```
    public String getItemName() {
        return itemName;
    }

    public double getCurrentBid() {
        return currentBid;
    }

    public void placeBid(double amount) {
        this.currentBid = amount;
    }

    @Override
    public String toString() {
        return "AuctionItem[Name=" + itemName + ", CurrentBid=" + currentBid + "]";
    }
}

// Concrete class for Physical Item
class PhysicalItem extends AuctionItem {
    private String condition;

    public PhysicalItem(String itemName, String condition) {
        super(itemName);
        this.condition = condition;
    }

    @Override
    public String toString() {
        return super.toString() + ", Condition=" + condition + "]";
```

```
    }
}

// User class
class User {
    private String username;
    private double balance;

    public User(String username, double balance) {
        this.username = username;
        this.balance = balance;
    }

    public String getUsername() {
        return username;
    }

    public double getBalance() {
        return balance;
    }

    public void updateBalance(double amount) {
        this.balance += amount;
    }
}

// Auction class
class Auction {
    private List<AuctionItem> items;
    private ConcurrentMap<String, User> users;
```

```
public Auction() {
    this.items = new ArrayList<>();
    this.users = new ConcurrentHashMap<>();
}

public void addItem(AuctionItem item) {
    items.add(item);
}

public void registerUser(User user) {
    users.put(user.getUsername(), user);
}

public void placeBid(String itemName, String username, double amount) {
    for (AuctionItem item : items) {
        if (item.getItemName().equals(itemName)) {
            if (users.containsKey(username)) {
                User user = users.get(username);
                if (amount > item.getCurrentBid() && amount <= user.getBalance()) {
                    item.placeBid(amount);
                    user.updateBalance(-amount);
                    System.out.println("Bid placed successfully.");
                } else {
                    System.out.println("Invalid bid amount or insufficient balance.");
                }
            } else {
                System.out.println("User not registered.");
            }
            return;
        }
    }
```

```
        System.out.println("Item not found.");
    }

    @Override
    public String toString() {
        return "Auction[Items=" + items + "]";
    }
}

// Main class to run the auction system
public class AuctionSystem {
    public static void main(String[] args) {
        Auction auction = new Auction();

        // Create some items and users
        PhysicalItem item1 = new PhysicalItem("Vintage Camera", "Good");
        User user1 = new User("Alice", 500.0);
        User user2 = new User("Bob", 300.0);

        auction.addItem(item1);
        auction.registerUser(user1);
        auction.registerUser(user2);

        // User places a bid
        auction.placeBid("Vintage Camera", "Alice", 200.0);
        System.out.println(auction);
    }
}
```