

Assignment: Abstraction (interface)

Part A: MCQ's

1. The main goal of abstraction as explained in the PDF is to:

- A. Hide variables
- B. Increase execution speed
- C. Reduce complexity by exposing only essential details
- D. Support polymorphism

Answer: C

2. Essential knowledge in abstraction refers to:

- A. Internal logic of method
- B. Constructor execution
- C. Method name, parameters, return type, and purpose
- D. JVM memory allocation

Answer: C

3. Which keyword is used to design an interface in Java?

- A. class
- B. abstract
- C. interface
- D. implements

Answer: C

4. According to the PDF, interface variables are:

- A. private and static
- B. public only
- C. public, static, and final
- D. protected and final

Answer: C

5. Why must interface variables be initialized?

- A. Because they are static
- B. Because they are final
- C. Because constructors initialize them
- D. Because JVM enforces it

Answer: B

6. Till JDK 1.7, interface methods were:

- A. Concrete
- B. Static
- C. Abstract only
- D. Default

Answer: C

7. Which statement about interface methods is TRUE?

- A. They can have method body
- B. They end with semicolon
- C. They must be protected
- D. They must be static

Answer: B

8. Interface methods are by default:

- A. private abstract
- B. public abstract
- C. protected
- D. final

Answer: B

9. Can we create an object of an interface?

- A. Yes
- B. No
- C. Only using new keyword
- D. Only in JVM

Answer: B

10. Which relationship is correct?

- A. Class extends interface
- B. Interface implements class
- C. Class implements interface
- D. Interface creates class

Answer: C

11. Java supports multiple inheritance using interface because:

- A. Interfaces have constructors
- B. Interfaces avoid ambiguity
- C. Interfaces use static methods
- D. JVM ignores conflicts

Answer: B

12. Why does interface not contain constructor?

- A. Constructors are private
- B. Interfaces cannot be instantiated
- C. No instance variables exist in interface
- D. JVM restriction

Answer: C

13. Which of the following is a marker interface?

- A. Runnable
- B. Serializable
- C. Comparable
- D. Callable

Answer: B

14. Marker interface is also called:

- A. Abstract interface
- B. Functional interface
- C. Tag interface
- D. Nested interface

Answer: C

15. Purpose of marker interface is to:

- A. Store data
- B. Perform inheritance
- C. Provide runtime information to JVM
- D. Increase security

Answer: C

16. Interface reference can refer to:

- A. Interface object
- B. Abstract class object
- C. Implemented class object
- D. Any class object

Answer: C

17. If reference is of interface type, accessible methods are:

- A. All class methods
- B. Only interface methods
- C. Only static methods
- D. Only default methods

Answer: B

18. Which statement correctly explains method access using an interface reference?

- A. Interface reference can access all methods of implementing class
- B. Interface reference can access only static methods
- C. Interface reference can access only methods declared in the interface
- D. Interface reference cannot call any method

Answer: C

19. Which concept is BEST demonstrated by ATM example?

- A. Encapsulation
- B. Abstraction
- C. Inheritance
- D. Polymorphism

Answer: B

by Kunal Sir

20. Interface acts like RBI because it:

- A. Controls execution
- B. Stores money
- C. Provides rules for implementation
- D. Creates objects

Answer: C



CJC
Complete Java Classes

Part B: Problem Statements

1. ATM Machine System

Problem Statement:

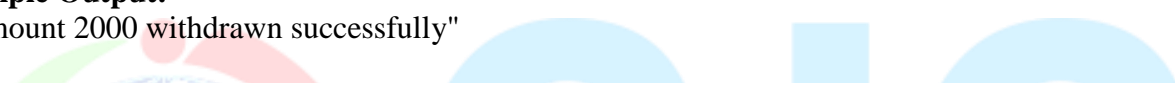
Design an `ATMService` interface that defines common ATM operations such as withdraw, deposit, and balance enquiry. Different banks can implement this interface according to their internal logic while the user interacts with a common ATM screen.

Sample Input:

Withdraw amount = 2000

Sample Output:

"Amount 2000 withdrawn successfully"



```
interface ATMService {
    void withdraw(int amount);
}
class SBIATM implements ATMService {
    public void withdraw(int amount) {
        System.out.println("SBI ATM: Amount " + amount + " withdrawn");
    }
}
class HDFCATM implements ATMService {
    public void withdraw(int amount) {
        System.out.println("HDFC ATM: Amount " + amount + " withdrawn");
    }
}
class ATMMain {
    public static void main(String[] args) {
        ATMService atm = new SBIATM();
        atm.withdraw(2000);
    }
}
```

2. Banking Rules System (RBI Example)

Problem Statement:

Create a `BankRules` interface that defines rules like minimum balance and interest rate. All banks must follow these rules while implementing their own banking services.

Sample Input:

Minimum Balance Check

Sample Output:

"Minimum balance rule applied"

```
interface BankRules {
    void followRules();
}
class SBI implements BankRules {
    public void followRules() {
        System.out.println("SBI follows RBI minimum balance rule");
    }
}
class ICICI implements BankRules {
    public void followRules() {
        System.out.println("ICICI follows RBI minimum balance rule");
    }
}
class BankMain {
    public static void main(String[] args) {
        BankRules bank = new SBI();
        bank.followRules();
    }
}
```

3. Payment System

Problem Statement:

Design a `Payment` interface with a `pay()` method. Implement it for Credit Card, UPI, and Net Banking payments so the system can switch payment modes dynamically.

Sample Input:

Payment Mode = UPI, Amount = 500

Sample Output:

"Payment of 500 completed using UPI"

```
interface Payment {
    void pay(int amount);
}
class CreditCardPayment implements Payment {
    public void pay(int amount) {
        System.out.println("Paid " + amount + " using Credit Card");
    }
}
class UPIPayment implements Payment {
    public void pay(int amount) {
        System.out.println("Paid " + amount + " using UPI");
    }
}
class NetBankingPayment implements Payment {
    public void pay(int amount) {
        System.out.println("Paid " + amount + " using Net Banking");
    }
}
class PaymentMain {
    public static void main(String[] args) {
        Payment payment = new UPIPayment();
        payment.pay(500);
    }
}
```


4. Notification System

Problem Statement:

Create a `Notification` interface to send alerts. Implement it for Email and SMS so the notification method can change without affecting business logic.

Sample Input:

Message = "Order Confirmed"

Sample Output:

"Email notification sent: Order Confirmed"

```
interface Notification {  
    void send(String message);  
}  
class EmailNotification implements Notification {  
    public void send(String message) {  
        System.out.println("Email sent: " + message);  
    }  
}  
class SMSNotification implements Notification {  
    public void send(String message) {  
        System.out.println("SMS sent: " + message);  
    }  
}  
class NotificationMain {  
    public static void main(String[] args) {  
        Notification notify = new EmailNotification();  
        notify.send("Order Confirmed");  
    }  
}
```

5. Shape Drawing Application

Problem Statement:

Design a `Shape` interface with a `draw()` method. Different shapes implement the interface to provide their own drawing logic.

Sample Input:

Shape = Circle

Sample Output:

"Drawing Circle"

```
interface Shape {
    void draw();
}
class Circle implements Shape {
    public void draw() {
        System.out.println("Drawing Circle");
    }
}
class Rectangle implements Shape {
    public void draw() {
        System.out.println("Drawing Rectangle");
    }
}
class ShapeMain {
    public static void main(String[] args) {
        Shape shape = new Circle();
        shape.draw();
    }
}
```

6. Vehicle System

Problem Statement:

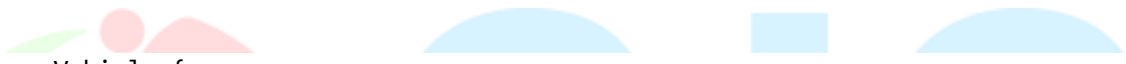
Create a `Vehicle` interface with a `start()` method. Implement it for `Car` and `Bike` to demonstrate common behavior with different implementations.

Sample Input:

Vehicle = Bike

Sample Output:

"Bike started"



```
interface Vehicle {  
    void start();  
}  
class Car implements Vehicle {  
    public void start() {  
        System.out.println("Car started");  
    }  
}  
class Bike implements Vehicle {  
    public void start() {  
        System.out.println("Bike started");  
    }  
}  
class VehicleMain {  
    public static void main(String[] args) {  
        Vehicle vehicle = new Bike();  
        vehicle.start();  
    }  
}
```

7. Database Connectivity

Problem Statement:

Design a `DBConnection` interface so the application can connect to different databases without changing core logic.

Sample Input:

Database = MySQL

Sample Output:

"Connected to MySQL Database"

```
interface DBConnection {
    void connect();
}
class MySQL implements DBConnection {
    public void connect() {
        System.out.println("Connected to MySQL Database");
    }
}
class OracleDB implements DBConnection {
    public void connect() {
        System.out.println("Connected to Oracle Database");
    }
}
class DBMain {
    public static void main(String[] args) {
        DBConnection db = new MySQL();
        db.connect();
    }
}
```

8. Printer Management System

Problem Statement:

Create a `Printer` interface with a `print()` method. Implement it for Laser and Inkjet printers.

Sample Input:

Document = Resume.pdf

Sample Output:

"Printing Resume.pdf using Laser Printer"

```
interface Printer {
    void print(String document);
}
class LaserPrinter implements Printer {
    public void print(String document) {
        System.out.println("Laser Printer printing: " + document);
    }
}
class InkjetPrinter implements Printer {
    public void print(String document) {
        System.out.println("Inkjet Printer printing: " + document);
    }
}
class PrinterMain {
    public static void main(String[] args) {
        Printer printer = new LaserPrinter();
        printer.print("Resume.pdf");
    }
}
```

9. File Handling System

Problem Statement:

Design a `FileOperation` interface for file operations such as read and write. Different file systems can implement it.

Sample Input:

Operation = Read

Sample Output:

"File read successfully"

```
interface FileOperation {  
    void read();  
}  
class TextFile implements FileOperation {  
    public void read() {  
        System.out.println("Reading Text File");  
    }  
}  
class PDFFile implements FileOperation {  
    public void read() {  
        System.out.println("Reading PDF File");  
    }  
}  
class FileMain {  
    public static void main(String[] args) {  
        FileOperation file = new TextFile();  
        file.read();  
    }  
}
```

10. Online Shopping Discount System

Problem Statement:

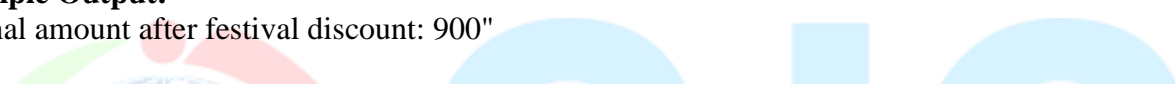
Create a `DiscountPolicy` interface to apply discounts. Different discount strategies can be implemented.

Sample Input:

Amount = 1000, Discount = Festival

Sample Output:

"Final amount after festival discount: 900"



```
interface DiscountPolicy {
    void applyDiscount(int amount);
}
class FestivalDiscount implements DiscountPolicy {
    public void applyDiscount(int amount) {
        System.out.println("Festival Discount Applied: " + (amount - 100));
    }
}
class SeasonalDiscount implements DiscountPolicy {
    public void applyDiscount(int amount) {
        System.out.println("Seasonal Discount Applied: " + (amount - 50));
    }
}
class DiscountMain {
    public static void main(String[] args) {
        DiscountPolicy discount = new FestivalDiscount();
        discount.applyDiscount(1000);
    }
}
```

11. Employee Salary Calculation

Problem Statement:

Design a `SalaryCalculator` interface for calculating salary of full-time and part-time employees.

Sample Input:

Employee Type = Part-Time, Hours = 20

Sample Output:

"Salary calculated: 4000"

```
interface SalaryCalculator {
    void calculateSalary();
}
class FullTimeEmployee implements SalaryCalculator {
    public void calculateSalary() {
        System.out.println("Full-Time Salary: 30000");
    }
}
class PartTimeEmployee implements SalaryCalculator {
    public void calculateSalary() {
        System.out.println("Part-Time Salary: 4000");
    }
}
class SalaryMain {
    public static void main(String[] args) {
        SalaryCalculator emp = new PartTimeEmployee();
        emp.calculateSalary();
    }
}
```

12. Login Authentication System

Problem Statement:


Create an `Authenticator` interface to authenticate users using password or OTP methods.

Sample Input:

Login Method = OTP

Sample Output:

"User authenticated using OTP"



```
interface Authenticator {  
    void authenticate();  
}  
class PasswordAuth implements Authenticator {  
    public void authenticate() {  
        System.out.println("Authenticated using Password");  
    }  
}  
class OTPAuth implements Authenticator {  
    public void authenticate() {  
        System.out.println("Authenticated using OTP");  
    }  
}  
class AuthMain {  
    public static void main(String[] args) {  
        Authenticator auth = new OTPAuth();  
        auth.authenticate();  
    }  
}
```

13. Media Player System

Problem Statement:

Design a `Playable` interface for playing audio and video files.

Sample Input:

Media Type = Audio

Sample Output:

"Playing audio file"

```
interface Playable {  
    void play();  
}  
class AudioPlayer implements Playable {  
    public void play() {  
        System.out.println("Playing Audio File");  
    }  
}  
class VideoPlayer implements Playable {  
    public void play() {  
        System.out.println("Playing Video File");  
    }  
}  
class MediaMain {  
    public static void main(String[] args) {  
        Playable media = new AudioPlayer();  
        media.play();  
    }  
}
```

14. Cloud Storage Service

Problem Statement:


Create a `StorageService` interface to upload files to different cloud platforms.

Sample Input:

File = data.txt

Sample Output:

"File uploaded to AWS Cloud"



```
interface StorageService {
    void upload(String file);
}
class AWSStorage implements StorageService {
    public void upload(String file) {
        System.out.println("Uploaded to AWS: " + file);
    }
}
class AzureStorage implements StorageService {
    public void upload(String file) {
        System.out.println("Uploaded to Azure: " + file);
    }
}
class CloudMain {
    public static void main(String[] args) {
        StorageService storage = new AWSStorage();
        storage.upload("data.txt");
    }
}
```

15. Logging Framework

Problem Statement:

Design a `Logger` interface to log messages to different destinations such as file or database.

Sample Input:

Log Message = "Login successful"

Sample Output:

"Log stored in file"

```
interface Logger {  
    void log(String message);  
}  
class FileLogger implements Logger {  
    public void log(String message) {  
        System.out.println("Log stored in File: " + message);  
    }  
}  
class DatabaseLogger implements Logger {  
    public void log(String message) {  
        System.out.println("Log stored in Database: " + message);  
    }  
}  
class LoggerMain {  
    public static void main(String[] args) {  
        Logger logger = new FileLogger();  
        logger.log("Login successful");  
    }  
}
```
