IT Skill Test GIC Myanmar

Duration: 30 Minutes *Total Questions: 8*

---

1. You are developing a simple banking application. You need to create a class to represent a bank account with methods for deposit and withdrawal. Which of the following implementations best adheres to OOP principles?

A.
```
public class BankAccount {
    public double balance;
    public void deposit(double amount) { balance += amount; }
    public void withdraw(double amount) { balance -= amount; }
}
```

B.
```
public class BankAccount {
    private double balance;
    public void deposit(double amount) {
        if (amount > 0) balance += amount;
    }
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) balance -= amount;
    }
}
```

C.
```
public class BankAccount {
    public static double balance;
    public static void deposit(double amount) { balance += amount; }
    public static void withdraw(double amount) { balance -= amount; }
}
```

D.
```
public class BankAccount {
    private double balance;
    public void setBalance(double newBalance) { balance = newBalance; }
    public double getBalance() { return balance; }
}
```

2. Which of the following is the correct way to define a constructor for a class named 'Employee' that takes a String parameter for the employee's name?

A. public void Employee(String name) { }

B. public Employee(String name) { }

C. Employee(String name) { }

D. void Employee(String name) { }

---

3. You're working on a project that involves processing customer orders. You need to implement a method that calculates the total price of an order, including tax. Which of the following method signatures is most appropriate?

A. public static void calculateTotalPrice(Order order, double taxRate)

B. public double calculateTotalPrice(Order order, double taxRate)

C. private double calculateTotalPrice(Order order, double taxRate)

D. public Order calculateTotalPrice(double taxRate)

4. In a team project, you notice that several classes are using similar code to validate user input. To improve code reusability, you decide to create a utility class for input validation. Which of the following is the best way to implement this utility class?

A.
```
public class InputValidator {
    public boolean validateEmail(String email) { /* validation logic */ }
    public boolean validatePhoneNumber(String phone) { /* validation logic */ }
}
```

B.
```
public class InputValidator {
    public static boolean validateEmail(String email) { /* validation logic */ }
    public static boolean validatePhoneNumber(String phone) { /* validation logic */ }
}
```

C.
```
public abstract class InputValidator {
    public abstract boolean validateEmail(String email);
    public abstract boolean validatePhoneNumber(String phone);
}
```

D.
```
public interface InputValidator {
    boolean validateEmail(String email);
    boolean validatePhoneNumber(String phone);
}
```

5. You're working on a class that represents a geometric shape. You want to ensure that the area calculation method is implemented by all subclasses, but the implementation will vary for each shape. How should you define this method in the base class?

A. public double calculateArea() { return 0; }

B. public abstract double calculateArea();

C. public final double calculateArea() { /* default implementation */ }

D. private double calculateArea() { /* default implementation */ }

6. You're implementing a method to find the first occurrence of a specific element in a large, unsorted array. The method should return the index of the element if found, or -1 if not found. Which looping construct would be most efficient?

A. for (int i = 0; i < array.length; i++) { ... }

B. while (i < array.length) { ... i++; }

C. do { ... i++; } while (i < array.length);

D. for (int element : array) { ... }

7. In a team project, you need to implement a logging system that ensures only one instance of the logger is created and used throughout the application. Which design pattern should you use, and how would you implement it in Java?

A.
```
public class Logger {
    private static Logger instance;
    private Logger() {}
    public static Logger getInstance() {
        if (instance == null) {
            instance = new Logger();
        }
        return instance;
    }
    public void log(String message) {
        System.out.println(message);
    }
}
```

B.
```
public class Logger {
    public Logger() {}
    public void log(String message) {
        System.out.println(message);
    }
}
```

C.
```
public class Logger {
    private Logger() {}
    public static Logger createInstance() {
        return new Logger();
    }
    public void log(String message) {
        System.out.println(message);
    }
}
```

D.
```
public abstract class Logger {
    public abstract void log(String message);
}
```

8. You are working on a feature that processes customer orders. The Order class has a status field that can be 'NEW', 'PROCESSING', 'SHIPPED', or 'DELIVERED'. You need to implement a method that updates the order status. Which of the following implementations is the most appropriate and adheres to OOP principles?

A.
```
public class Order {
    private String status;
    public void setStatus(String newStatus) {
        status = newStatus;
    }
}
```

B.
```
public class Order {
    private enum Status { NEW, PROCESSING, SHIPPED, DELIVERED }
    private Status status;
    public void setStatus(Status newStatus) {
        status = newStatus;
    }
}
```

C.
```
public class Order {
    public static final int NEW = 0;
    public static final int PROCESSING = 1;
    public static final int SHIPPED = 2;
    public static final int DELIVERED = 3;
    private int status;
    public void setStatus(int newStatus) {
        status = newStatus;
    }
}
```

D.
```
public class Order {
    private String status;
    public void setStatus(String newStatus) {
        if (newStatus.equals("NEW") || newStatus.equals("PROCESSING") ||
            newStatus.equals("SHIPPED") || newStatus.equals("DELIVERED")) {
            status = newStatus;
        }
    }
}
```