

Singleton Design Pattern

1. What will be the output of the following Singleton pattern implementation?

```
class Singleton {
    private static Singleton instance;

    private Singleton() { }

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    public void showMessage() {
        System.out.println("Singleton Instance");
    }
}

public class Test {
    public static void main(String[] args) {
        Singleton s1 = Singleton.getInstance();
        Singleton s2 = Singleton.getInstance();
        s1.showMessage();
        System.out.println(s1 == s2);
    }
}
```

(Single Correct Answer)

- A) Singleton Instance true
- B) Singleton Instance false
- C) Compilation error
- D) Runtime error

2. Which of the following are true about the Singleton pattern?

```
class Singleton {
    private static Singleton instance = new Singleton();

    private Singleton() { }

    public static Singleton getInstance() {
        return instance;
    }
}
```

- A) It ensures that only one instance of the class is created.

- B) It uses lazy initialization.
- C) It is thread-safe in this implementation.
- D) It can be subclassed to create multiple instances.

Factory Design Pattern

3. What will be the output of the following Factory pattern implementation?

```
interface Product {
    void create();
}

class ConcreteProductA implements Product {
    public void create() { System.out.println("Product A"); }
}

class ConcreteProductB implements Product {
    public void create() { System.out.println("Product B"); }
}

class ProductFactory {
    public static Product getProduct(String type) {
        if (type.equals("A")) return new ConcreteProductA();
        if (type.equals("B")) return new ConcreteProductB();
        return null;
    }
}

public class Test {
    public static void main(String[] args) {
        Product p1 = ProductFactory.getProduct("A");
        Product p2 = ProductFactory.getProduct("B");
        p1.create();
        p2.create();
    }
}
```

(Single Correct Answer)

- A) Product A Product B
 - B) Product B Product A
 - C) Product A null
 - D) Product B null
- ### 4. Which of the following statements are true about the Factory design pattern?

```
interface Animal {
```

```

        void speak();
    }

    class Dog implements Animal {
        public void speak() { System.out.println("Woof"); }
    }

    class Cat implements Animal {
        public void speak() { System.out.println("Meow"); }
    }

    class AnimalFactory {
        public static Animal getAnimal(String type) {
            if (type.equals("Dog")) return new Dog();
            if (type.equals("Cat")) return new Cat();
            return null;
        }
    }

```

- A) The Factory pattern helps in creating objects without specifying the exact class.
- B) The Factory pattern is not used for creating objects of different types.
- C) It uses a static method to create objects.
- D) The Factory pattern can return `null` if the type is not recognized.

Abstract Factory Design Pattern

5. What will be the output of the following Abstract Factory pattern implementation?

```

interface Animal {
    void makeSound();
}

class Dog implements Animal {
    public void makeSound() { System.out.println("Bark"); }
}

class Cat implements Animal {
    public void makeSound() { System.out.println("Meow"); }
}

interface AnimalFactory {
    Animal createAnimal();
}

class DogFactory implements AnimalFactory {
    public Animal createAnimal() { return new Dog(); }
}

class CatFactory implements AnimalFactory {
    public Animal createAnimal() { return new Cat(); }
}

```

```

public class Test {
    public static void main(String[] args) {
        AnimalFactory factory = new DogFactory();
        Animal animal = factory.createAnimal();
        animal.makeSound();
    }
}

```

(Single Correct Answer)

- A) Bark
- B) Meow
- C) Compilation error
- D) Runtime error

6. Which of the following statements are true about the Abstract Factory design pattern?

```

interface Furniture {
    void create();
}

class Chair implements Furniture {
    public void create() { System.out.println("Chair created"); }
}

class Table implements Furniture {
    public void create() { System.out.println("Table created"); }
}

interface FurnitureFactory {
    Furniture createFurniture();
}

class ChairFactory implements FurnitureFactory {
    public Furniture createFurniture() { return new Chair(); }
}

class TableFactory implements FurnitureFactory {
    public Furniture createFurniture() { return new Table(); }
}

```

- A) It provides an interface for creating families of related or dependent objects.
- B) It allows the client to create objects without specifying the concrete classes.
- C) It can use a single factory to create multiple objects of different types.
- D) It is used for creating objects of a single type.

Builder Design Pattern

7. What will be the output of the following Builder pattern implementation?

```
class Computer {
    private String CPU;
    private String RAM;
    private String storage;

    public void setCPU(String CPU) { this.CPU = CPU; }
    public void setRAM(String RAM) { this.RAM = RAM; }
    public void setStorage(String storage) { this.storage = storage; }
}

@Override
public String toString() {
    return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", storage="
+ storage + "]\n";
}

class ComputerBuilder {
    private Computer computer;

    public ComputerBuilder() { computer = new Computer(); }

    public ComputerBuilder setCPU(String CPU) { computer.setCPU(CPU);
return this; }
    public ComputerBuilder setRAM(String RAM) { computer.setRAM(RAM);
return this; }
    public ComputerBuilder setStorage(String storage) {
computer.setStorage(storage); return this; }

    public Computer build() { return computer; }
}

public class Test {
    public static void main(String[] args) {
        Computer comp = new ComputerBuilder()
            .setCPU("Intel")
            .setRAM("16GB")
            .setStorage("512GB SSD")
            .build();

        System.out.println(comp);
    }
}
```

(Single Correct Answer)

- A) Computer [CPU=Intel, RAM=16GB, storage=512GB SSD]
- B) Computer [CPU=Intel, RAM=16GB, storage=null]
- C) Compilation error
- D) Runtime error

8. Which of the following statements are true about the Builder design pattern?

```
class House {
    private String walls;
    private String roof;
    private String windows;

    public void setWalls(String walls) { this.walls = walls; }
    public void setRoof(String roof) { this.roof = roof; }
    public void setWindows(String windows) { this.windows = windows; }
}

class HouseBuilder {
    private House house;

    public HouseBuilder() { house = new House(); }

    public HouseBuilder buildWalls(String walls) {
        house.setWalls(walls); return this; }
    public HouseBuilder buildRoof(String roof) { house.setRoof(roof);
        return this; }
    public HouseBuilder buildWindows(String windows) {
        house.setWindows(windows); return this; }

    public House build() { return house; }
}
```

- A) The Builder pattern separates the construction of a complex object from its representation.
- B) The Builder pattern provides a fluent interface.
- C) The Builder pattern can be used to create immutable objects.
- D) The Builder pattern is used for creating objects with a single constructor.

Template Method Design Pattern

9. What will be the output of the following Template Method pattern implementation?

```
abstract class AbstractClass {
    public final void templateMethod() {
        step1();
        step2();
        step3();
    }

    abstract void step1();
    abstract void step2();

    void step3() { System.out.println("Step 3"); }
}
```

```

    }

    class ConcreteClass extends AbstractClass {
        void step1() { System.out.println("Step 1"); }
        void step2() { System.out.println("Step 2"); }
    }

    public class Test {
        public static void main(String[] args) {
            AbstractClass obj = new ConcreteClass();
            obj.templateMethod();
        }
    }

```

(Single Correct Answer)

- A) Step 1 Step 2 Step 3
- B) Step 1 Step 3
- C) Step 2 Step 3
- D) Compilation error

10. Which of the following statements are true about the Template Method design pattern?

```

abstract class Meal {
    public final void prepareMeal() {
        cook();
        serve();
    }

    abstract void cook();
    abstract void serve();
}

class Breakfast extends Meal {
    void cook() { System.out.println("Cooking Breakfast"); }
    void serve() { System.out.println("Serving Breakfast"); }
}

class Dinner extends Meal {
    void cook() { System.out.println("Cooking Dinner"); }
    void serve() { System.out.println("Serving Dinner"); }
}

```

- A) It defines the skeleton of an algorithm in a base class but lets subclasses override specific steps.
- B) It allows subclasses to redefine certain steps of an algorithm without changing the algorithm's structure.
- C) It is not suitable for scenarios where an algorithm's steps are fixed.

D) It provides a way to encapsulate the algorithm in a concrete subclass.

Bridge Design Pattern

11. What will be the output of the following Bridge pattern implementation?

```
interface Implementor {
    void operation();
}

class ConcreteImplementorA implements Implementor {
    public void operation() {
        System.out.println("ConcreteImplementorA Operation");
    }
}

class ConcreteImplementorB implements Implementor {
    public void operation() {
        System.out.println("ConcreteImplementorB Operation");
    }
}

abstract class Abstraction {
    protected Implementor implementor;

    public Abstraction(Implementor implementor) {
        this.implementor = implementor;
    }

    public abstract void performOperation();
}

class RefinedAbstraction extends Abstraction {
    public RefinedAbstraction(Implementor implementor) {
        super(implementor);
    }

    public void performOperation() {
        implementor.operation();
    }
}

public class Test {
    public static void main(String[] args) {
        Abstraction ab = new RefinedAbstraction(new
        ConcreteImplementorA());
        ab.performOperation();
    }
}
```

(Single Correct Answer)

- A) ConcreteImplementorA Operation
- B) ConcreteImplementorB Operation
- C) Operation

D) Compilation error

12. Which of the following statements are true about the Bridge design pattern?

```
interface DrawingAPI {
    void drawCircle(double x, double y, double radius);
}

class DrawingAPI1 implements DrawingAPI {
    public void drawCircle(double x, double y, double radius) {
        System.out.println("API1.circle at " + x + ":" + y + " radius " + radius);
    }
}

class DrawingAPI2 implements DrawingAPI {
    public void drawCircle(double x, double y, double radius) {
        System.out.println("API2.circle at " + x + ":" + y + " radius " + radius);
    }
}

abstract class CircleShape {
    protected DrawingAPI drawingAPI;

    protected CircleShape(DrawingAPI drawingAPI) {
        this.drawingAPI = drawingAPI;
    }

    public abstract void draw();
    public abstract void resizeByPercentage(double pct);
}

class CircleShapeImpl extends CircleShape {
    private double x, y, radius;

    protected CircleShapeImpl(double x, double y, double radius, DrawingAPI drawingAPI) {
        super(drawingAPI);
        this.x = x;
        this.y = y;
        this.radius = radius;
    }

    public void draw() {
        drawingAPI.drawCircle(x, y, radius);
    }

    public void resizeByPercentage(double pct) {
        radius *= (1 + pct / 100);
    }
}
```

A) The Bridge pattern separates abstraction from implementation.

B) The Bridge pattern is used to create a bridge between two unrelated classes.

C) It allows changing the implementation of an abstraction without changing the abstraction itself.

D) It uses inheritance to extend class functionality.

Proxy Design Pattern

13. What will be the output of the following Proxy pattern implementation?

```
interface Image {
    void display();
}

class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadImageFromDisk();
    }

    private void loadImageFromDisk() {
        System.out.println("Loading " + filename);
    }

    public void display() {
        System.out.println("Displaying " + filename);
    }
}

class ProxyImage implements Image {
    private RealImage realImage;
    private String filename;

    public ProxyImage(String filename) {
        this.filename = filename;
    }

    public void display() {
        if (realImage == null) {
            realImage = new RealImage(filename);
        }
        realImage.display();
    }
}

public class Test {
    public static void main(String[] args) {
        Image image = new ProxyImage("test_image.jpg");
        image.display();
    }
}
```

(Single Correct Answer)

- A) Loading test_image.jpg Displaying test_image.jpg
- B) Displaying test_image.jpg
- C) Loading test_image.jpg
- D) Compilation error

14. Which of the following statements are true about the Proxy design pattern?

```
interface RealObject {
    void performAction();
}

class RealObjectImpl implements RealObject {
    public void performAction() {
        System.out.println("Action performed");
    }
}

class ProxyObject implements RealObject {
    private RealObject realObject;

    public void performAction() {
        if (realObject == null) {
            realObject = new RealObjectImpl();
        }
        realObject.performAction();
    }
}
```

- A) The Proxy pattern controls access to an object.
- B) The Proxy pattern can perform additional actions before or after delegating to the real object.
- C) It is used to create multiple instances of an object.
- D) It helps in lazy initialization and access control.

Creating Immutable Classes

15. What will be the output of the following immutable class implementation?

```
final class ImmutableClass {
    private final int value;

    public ImmutableClass(int value) {
        this.value = value;
    }
}
```

```

        public int getValue() {
            return value;
        }
    }

    public class Test {
        public static void main(String[] args) {
            ImmutableClass obj = new ImmutableClass(10);
            System.out.println(obj.getValue());
        }
    }

```

(Single Correct Answer)

- A) 10
- B) Compilation error
- C) Runtime error
- D) null

16. Which of the following statements are true about immutable classes?

```

final class Immutable {
    private final String data;

    public Immutable(String data) {
        this.data = data;
    }

    public String getData() {
        return data;
    }
}

```

- A) Immutable objects cannot be modified after they are created.
- B) Immutable classes must be declared as `final`.
- C) All fields in an immutable class should be `final`.
- D) Immutable objects are created using `setter` methods.

8 Features

17. What will be the output of the following Lambda expression implementation?

```

interface StringManipulator {

```

```

        String manipulate(String str);
    }

    public class Test {
        public static void main(String[] args) {
            StringManipulator manipulator = s -> s.toUpperCase();
            System.out.println(manipulator.manipulate(""));
        }
    }

```

(Single Correct Answer)

A) JAVA

B) java

C) Compilation error

D) Runtime error

18. Which of the following are true about Lambda expressions?

```

@FunctionalInterface
interface Calculator {
    int compute(int a, int b);
}

public class Test {
    public static void main(String[] args) {
        Calculator add = (a, b) -> a + b;
        Calculator multiply = (a, b) -> a * b;
        System.out.println(add.compute(5, 3));
        System.out.println(multiply.compute(5, 3));
    }
}

```

A) Lambda expressions can be used to implement functional interfaces.

B) Lambda expressions can have multiple methods.

C) Lambda expressions can be used to simplify code that requires an implementation of an interface with a single abstract method.

D) Lambda expressions are anonymous functions.

Working with the Date/Time API

19. What will be the output of the following code using 8 Date/Time API?

```

import .time.LocalDate;
import .time.format.DateTimeFormatter;

```

```

public class Test {
    public static void main(String[] args) {
        LocalDate date = LocalDate.now();
        DateTimeFormatter formatter =
        DateTimeFormatter.ofPattern("dd/MM/yyyy");
        System.out.println(date.format(formatter));
    }
}

```

(Single Correct Answer)

- A) Current date in dd/MM/yyyy format
- B) yyyy/MM/dd format
- C) dd-MM-yyyy format
- D) Compilation error

20. Which of the following statements are true about 8 Date/Time API?

```

import .time.LocalDateTime;
import .time.ZoneId;
import .time.ZonedDateTime;

public class Test {
    public static void main(String[] args) {
        LocalDateTime localDateTime = LocalDateTime.now();
        ZonedDateTime zonedDateTime =
        localDateTime.atZone(ZoneId.of("America/New_York"));
        System.out.println(zonedDateTime);
    }
}

```

- A) The Date/Time API provides immutable classes for date and time.
- B) The Date/Time API allows working with time zones using `ZoneId`.
- C) The Date/Time API supports mutable date and time objects.
- D) The `ZonedDateTime` class includes date, time, and time zone information.

Generic Classes

21. What will be the output of the following code snippet with generic classes?

```

class Box<T> {

```

```

    private T content;

    public void setContent(T content) {
        this.content = content;
    }

    public T getContent() {
        return content;
    }
}

public class Test {
    public static void main(String[] args) {
        Box<String> stringBox = new Box<>();
        stringBox.setContent("Hello");
        System.out.println(stringBox.getContent());
    }
}

```

(Single Correct Answer)

- A) Hello
- B) null
- C) Compilation error
- D) Runtime error

22. Which of the following statements are true about generic classes?

```

class GenericClass<T> {
    private T value;

    public GenericClass(T value) {
        this.value = value;
    }

    public T getValue() {
        return value;
    }
}

```

- A) Generics allow type safety without needing to cast objects.
- B) Generics can be used with classes, interfaces, and methods.
- C) Generics support primitive types directly.
- D) Generics are implemented through type erasure at runtime.

23. What is the output of the following wildcard usage with generics?

```
import .util.ArrayList;
import .util.List;

public class Test {
    public static void main(String[] args) {
        List<? extends Number> numbers = new ArrayList<Integer>();
        // numbers.add(1); // This line would cause a compilation
error      System.out.println(numbers);
    }
}
```

(Single Correct Answer)

- A) Compilation error
- B) Empty list
- C) Runtime error
- D) No output

24. Which of the following statements are true about wildcard parameter types in generics?

```
import .util.List;

public class Test {
    public static void printList(List<?> list) {
        for (Object obj : list) {
            System.out.println(obj);
        }
    }
}
```

- A) List<?> represents a list of unknown type.
- B) Wildcards can be used to specify upper and lower bounds in generics.
- C) Wildcards provide a way to restrict the types that can be used as type arguments.
- D) Wildcards allow adding elements to a list.

25. What is the output of the following code snippet using bounded wildcards?

```
import .util.ArrayList;
import .util.List;
```



```
public class Test {  
    public static void printNumbers(List<? extends Number> numbers) {  
        for (Number number : numbers) {  
            System.out.println(number);  
        }  
    }  
  
    public static void main(String[] args) {  
        List<Integer> integers = new ArrayList<>();  
        integers.add(1);  
        integers.add(2);  
        printNumbers(integers);  
    }  
}
```

(Single Correct Answer)

- A) 1 2
- B) Compilation error
- C) No output
- D) Runtime error