**Quiz 1**

**1. Which of the following is true about Java's final keyword?**

a) final can be applied to classes, methods, and variables.

b) final classes can be subclassed.

c) final methods cannot be inherited by subclasses.

d) final variables can be changed once initialized.

Answer: a) final can be applied to classes, methods, and variables.

Explanation: The final keyword in Java is used to restrict the user. It can be applied to classes (to prevent inheritance), methods (to prevent method overriding), and variables (to make them constants). Option b) is incorrect because final classes cannot be subclassed. Option c) is incorrect because final methods cannot be overridden, not inherited. Option d) is incorrect because final variables can only be assigned once and cannot be changed thereafter.

**2. What will be the output of the following code?**

public class Main {

public static void main(String[] args) {

int a = 5;

int b = a++;

System.out.println(a + " " + b);

}

}

a) 4 5

b) 5 5

c) 6 5

d) 5 6

Answer: c) 6 5

Explanation: The a++ operation is a postincrement operator. It means the value of a is assigned to b before a is incremented. Therefore, b gets the value 5, and then a is incremented to 6. Thus, the output is 6 5.

**3. In Java, which of the following statements is correct about exception handling?**

a) Checked exceptions are subclasses of RuntimeException.

b) Unchecked exceptions are not subject to the throws clause.

c) A finally block is always executed, regardless of an exception.

d) The throw keyword is used to declare exceptions in a method signature.

Answer: c) A finally block is always executed, regardless of an exception.

Explanation: The finally block in Java is always executed after the try block, regardless of whether an exception is thrown or not. Option a) is incorrect because checked exceptions are not subclasses of RuntimeException; they are subclasses of Exception. Option b) is incorrect because unchecked exceptions (subclasses of RuntimeException) are also not subject to the throws clause. Option d) is incorrect because the throws keyword is used in a method signature to declare exceptions, not throw.

**4. What is the primary purpose of the Optional class in Java 8?**

a) To replace null values with a valid default.

b) To enable null checking at runtime.

c) To prevent NullPointerException by wrapping nullable types.

d) To allow lazy initialization of objects.

Answer: c) To prevent NullPointerException by wrapping nullable types.

Explanation: The Optional class in Java 8 is used to represent a value that may or may not be present, thus preventing NullPointerException. Option a) is incorrect because Optional doesn’t replace null values but wraps them. Option b) is incorrect because Optional provides compiletime null checking, not runtime. Option d) is incorrect because Optional is not used for lazy initialization.

**5. Given the following stream operation, what will be the output?**

List<String> names = Arrays.asList("John", "Paul", "George", "Ringo");

String result = names.stream()

.filter(s > s.startsWith("P"))

.findAny()

.orElse("No match");

System.out.println(result);

a) John

b) Paul

c) No match

d) Ringo

Answer: b) Paul

Explanation: The filter method filters the stream to include only names that start with "P". The findAny method returns any element that matches the filter. Since "Paul" is the only name starting with "P", the output is "Paul".

**6. Which of the following is not a functional interface in Java 8?**

a) Runnable

b) Comparator

c) Callable

d) Serializable

Answer: d) Serializable

Explanation: A functional interface in Java 8 is an interface with exactly one abstract method. Runnable, Comparator, and Callable are functional interfaces, while Serializable is not because it does not have any abstract methods.

**7. How does the forEach method in Java 8 differ from a traditional for loop?**

a) forEach can only be used with lists.

b) forEach does not support parallel processing.

c) forEach can be used with streams for better performance.

d) forEach requires explicit indexing to access elements.

Answer: c) forEach can be used with streams for better performance.

Explanation: The forEach method is part of the Stream API and can be used to process elements in a stream, potentially in parallel, providing better performance. Option a) is incorrect because forEach can be used with any Iterable. Option b) is incorrect because forEach can be used in parallel streams. Option d) is incorrect because forEach does not require explicit indexing.

**8. Which of the following code snippets correctly uses the map function to convert a list of integers to their squares?**

List<Integer> numbers = Arrays.asList(1, 2, 3, 4);

a) numbers.stream().map(i > i i).collect(Collectors.toList());

b) numbers.map(i > i i).collect(Collectors.toList());

c) numbers.forEach(i > i i).collect(Collectors.toList());

d) numbers.stream().forEach(i > i i).collect(Collectors.toList());

Answer: a) numbers.stream().map(i > i i).collect(Collectors.toList());

Explanation: To convert each element of the list to its square, map function is used after converting the list to a stream. Option a) correctly maps each integer to its square and collects the results into a list. Option b) is incorrect because map should be used on a stream, not directly on a list. Option c) is incorrect because forEach is used for sideeffects, not for transformations. Option d) is incorrect because forEach does not return a result.

**9. You are developing a system that requires creating complex objects step by step. Which design pattern would be most appropriate?**

a) Factory Pattern

b) Builder Pattern

c) Prototype Pattern

d) Singleton Pattern

Answer: b) Builder Pattern

Explanation: The Builder Pattern is used to construct complex objects step by step. It allows for the creation of objects with a lot of optional parameters in a controlled manner. Option a) is incorrect because the Factory Pattern is used to create objects without specifying the exact class. Option c) is incorrect because the Prototype Pattern is used to create objects by copying an existing instance. Option d) is incorrect because the Singleton Pattern ensures a single instance of a class.

**10. In a system where a single instance of a class is needed to coordinate actions across the system, which pattern would you choose?**

a) Singleton Pattern

b) Observer Pattern

c) Adapter Pattern

d) Strategy Pattern

Answer: a) Singleton Pattern

Explanation: The Singleton Pattern ensures that only one instance of a class exists and provides a global point of access to that instance. Option b) is incorrect because the Observer Pattern is used for notifying dependent objects of changes. Option c) is incorrect because the Adapter Pattern is used to make incompatible interfaces compatible. Option d) is incorrect because the Strategy Pattern is used to define a family of algorithms.

**11. A system needs to switch between different algorithms or strategies at runtime. Which pattern should be used?**

a) Strategy Pattern

b) Observer Pattern

c) Factory Pattern

d) Composite Pattern

Answer: a) Strategy Pattern

Explanation: The Strategy Pattern is used to define a family of algorithms, encapsulate each one, and make them interchangeable. This allows the algorithm to be selected at runtime. Option b) is incorrect because the Observer Pattern is used to notify observers of changes. Option c) is incorrect because the Factory Pattern is used to create objects. Option d) is incorrect because the Composite Pattern is used to treat individual objects and compositions of objects uniformly.

**12. You need to decouple an abstraction from its implementation so that the two can vary independently. Which pattern would you use?**

a) Bridge Pattern

b) Decorator Pattern

c) Adapter Pattern

d) Facade Pattern

Answer: a) Bridge Pattern

Explanation: The Bridge Pattern is used to decouple an abstraction from its implementation so that both can vary independently. Option b) is incorrect because the Decorator Pattern is used to add behavior to objects dynamically. Option c) is incorrect because the Adapter Pattern is used to convert one interface into another. Option d) is incorrect because the Facade Pattern provides a simplified interface to a complex subsystem.

**13. Which SOLID principle is violated if a class has multiple responsibilities?**

a) Single Responsibility Principle

b) Open/Closed Principle

c) Liskov Substitution Principle

d) Interface Segregation Principle

Answer: a) Single Responsibility Principle

Explanation: The Single Responsibility Principle states that a class should have only one reason to change, meaning it should have only one responsibility. Option b) is incorrect because the Open/Closed Principle states that classes should be open for extension but closed for modification. Option c) is incorrect because the Liskov Substitution Principle ensures that subclasses can be substituted for their base classes. Option d) is incorrect because the Interface Segregation Principle states that clients should not be forced to depend on interfaces they do not use.

**14. The Open/Closed Principle states that:**

a) Classes should be open for extension but closed for modification.

b) Classes should be open for modification but closed for extension.

c) Methods should not be overloaded.

d) A class should have only one reason to change.

Answer: a) Classes should be open for extension but closed for modification.

Explanation: The Open/Closed Principle states that a class should be open for extension (new functionality) but closed for modification (existing functionality should not be altered). Option b) is incorrect because it contradicts the principle. Option c) is incorrect because method overloading is unrelated to the Open/Closed Principle. Option d) describes the Single Responsibility Principle.

**15. Which SOLID principle encourages the use of interfaces over concrete classes?**

a) Dependency Inversion Principle

b) Liskov Substitution Principle

c) Interface Segregation Principle

d) Open/Closed Principle

Answer: a) Dependency Inversion Principle

Explanation: The Dependency Inversion Principle encourages depending on abstractions (interfaces) rather than concrete implementations to reduce coupling and increase flexibility. Option b) is incorrect because the Liskov Substitution Principle ensures that subclasses can be substituted for their base classes. Option c) is incorrect because the Interface Segregation Principle advocates creating small, specific interfaces rather than one large interface. Option d) is incorrect because the Open/Closed Principle deals with extending functionality without modifying existing code.

**16. Which Java 8 feature can be used to implement a Singleton design pattern in a threadsafe manner?**

a) Streams

b) Optional

c) Lambda Expressions

d) Enum

Answer: d) Enum

Explanation: Enums in Java are inherently threadsafe and provide a simple way to implement the Singleton design pattern. They ensure that only one instance of the enum is created. Options a), b), and c) are incorrect because Streams, Optional, and Lambda Expressions are not related to implementing the Singleton pattern.

**17. Given a list of integers, write a Java 8 stream operation to return the sum of all even numbers:**

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);

a) numbers.stream().filter(i > i % 2 == 0).mapToInt(Integer::intValue).sum();

b) numbers.stream().filter(i > i % 2 == 0).sum();

c) numbers.forEach(i > i % 2 == 0).sum();

d) numbers.filter(i > i % 2 == 0).mapToInt(Integer::intValue).sum();

Answer: a) numbers.stream().filter(i > i % 2 == 0).mapToInt(Integer::intValue).sum();

Explanation: Option a) correctly filters even numbers, maps them to int values, and sums them. Option b) is incorrect because the sum() method does not exist directly on a stream of integers without mapping. Option c) is incorrect because forEach is used for sideeffects, not for transformations or reductions. Option d) is incorrect because filter should be used with stream().

**18. Which design pattern would you use to create a series of related objects that follow a common interface?**

a) Factory Pattern

b) Prototype Pattern

c) Singleton Pattern

d) Builder Pattern

Answer: a) Factory Pattern

Explanation: The Factory Pattern provides an interface for creating objects without specifying the exact class. It is used to create a series of related objects that implement a common interface. Options b), c), and d) are incorrect because the Prototype Pattern is for copying objects, the Singleton Pattern ensures a single instance, and the Builder Pattern is for creating complex objects step by step.

**19. How can you use Java 8 features to implement a Chain of Responsibility pattern?**

a) Use lambdas to represent handlers in the chain.

b) Use streams to sequentially process each handler.

c) Use Optional to handle the absence of a handler.

d) Use method references to dynamically link handlers.

Answer: a) Use lambdas to represent handlers in the chain.

Explanation: Lambdas can be used to represent handlers in a Chain of Responsibility pattern, allowing for a flexible chain of operations. Options b), c), and d) are incorrect because streams, Optional, and method references do not directly implement the Chain of Responsibility pattern.

**20. Which of the following implementations ensures that the Singleton pattern is threadsafe and provides lazy initialization in Java?**

**Options:**

**A)**

public class Singleton {

private static Singleton instance;

private Singleton() {}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

}

**B)**

public class Singleton {

private static Singleton instance = new Singleton();

private Singleton() {}

public static Singleton getInstance() {

return instance;

}

}

**C)**

public class Singleton {

private static Singleton instance;

private Singleton() {}

public static synchronized Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

}

**D)**

public class Singleton {

private static class SingletonHelper {

private static final Singleton INSTANCE = new Singleton();

}

private Singleton() {}

public static Singleton getInstance() {

return SingletonHelper.INSTANCE;

}

}

**Correct Answer:** D) The implementation using the static inner class (SingletonHelper) ensures thread safety and lazy initialization without requiring synchronized blocks, making it the most efficient option.