Singleton Design Pattern

1. **Which of the following correctly implements the Singleton design pattern in ?**

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

A) Correct implementation
B) Incorrect because `instance` should be `final`
C) Incorrect because `getInstance()` should be synchronized
D) Incorrect because the constructor should be public

Factory Design Pattern

2. **Which method is characteristic of the Factory Design Pattern?**

```
public class ShapeFactory {
    public Shape getShape(String shapeType) {
        if (shapeType == null) {
            return null;
        }
        if (shapeType.equalsIgnoreCase("CIRCLE")) {
            return new Circle();
        } else if (shapeType.equalsIgnoreCase("RECTANGLE")) {
            return new Rectangle();
        } else if (shapeType.equalsIgnoreCase("SQUARE")) {
            return new Square();
        }
        return null;
    }
}
```

A) `getShape(String shapeType)`
B) `Shape` interface
C) `Circle`, `Rectangle`, `Square` classes
D) `main()` method

Abstract Factory Design Pattern

3. **Identify the correct class for an Abstract Factory Design Pattern:**

```
public abstract class AbstractFactory {
    abstract Color getColor(String color);
    abstract Shape getShape(String shape);
}
```

A) FactoryProducer
B) ShapeFactory
C) AbstractFactory
D) ColorFactory

Builder Design Pattern

4. **What is the purpose of the Builder Design Pattern?** A) To simplify the creation of complex objects
B) To ensure only one instance of a class exists
C) To create objects from a pool
D) To provide an interface for creating families of related objects

Template Method Design Pattern

5. **Which method is the template method in the following class?**

```
public abstract class Game {
    abstract void initialize();
    abstract void startPlay();
    abstract void endPlay();

    public final void play() {
        initialize();
        startPlay();
        endPlay();
    }
}
```

A) `initialize()`
B) `startPlay()`
C) `endPlay()`
D) `play()`

Bridge Design Pattern

6. **In the Bridge Design Pattern, what role does the `Bridge` interface play?** A) It is the base class for all objects
B) It separates the abstraction from the implementation
C) It defines the implementation
D) It implements the bridge interface

Proxy Design Pattern

7. **What is the purpose of the Proxy Design Pattern?** A) To create objects from a pool
   B) To provide a placeholder for another object to control access to it
   C) To ensure only one instance of a class exists
   D) To create families of related objects

Creating Immutable Classes

8. **Which statement is true about immutable classes in ?** A) They can have setter methods
   B) Their fields can be modified after construction
   C) They can be extended
   D) They do not allow modification after construction

8 Features - Lambdas

9. **What is the motivation for introducing Lambda expressions in 8?** A) To improve code readability and conciseness
   B) To allow multiple inheritance
   C) To improve security
   D) To enforce strict typing
10. **What is the syntax for a simple lambda expression that takes two integers and returns their sum?**

```
A) (int a, int b) -> a + b
B) (a, b) -> return a + b;
C) (int a, int b) { a + b }
D) (int a, int b) -> { return a + b; }
```

8 Features - Functional Interfaces

11. **Which of the following is a functional interface?** A) `Runnable`
    B) `Comparator`
    C) `Callable`
    D) All of the above

8 Features - Method References

12. **How would you write a method reference for the static method `parseInt` of `Integer` class?**

    A) `Integer::parseInt`
    B) `Integer.parseInt`
    C) `Integer->parseInt`
    D) `Integer::new`

Working with Date/Time API

13. **How do you create an instance of `LocalDate` representing the current date?**

```
A) LocalDate.now()
B) LocalDate.ofNow()
C) LocalDate.current()
D) new LocalDate()
```

14. **What does the following code snippet return?**

```
LocalTime time = LocalTime.of(10, 30);
```

A) Time set to 10:30 AM
B) Time set to 10:30 PM
C) Current time
D) Invalid time

15. **Which class should you use to represent a date and time with time zone information?** A) `LocalDateTime`
B) `LocalTime`
C) `ZonedDateTime`
D) `Instant`

Generic Classes

16. **How do you define a generic class in ?**

```
A) public class Box<T> {}
B) public class <T> Box {}
C) public <T> class Box {}
D) public class Box <> {}
```

17. **Which wildcard represents an upper bounded wildcard?** A) `<? super T>`
B) `<T extends ?>`
C) `<?>`
D) `<? extends T>`

Collections Framework

18. **How do you create a list of strings using type inference diamond?**

```
A) List<String> list = new ArrayList<>();
B) List<String> list = new ArrayList<String>();
C) List<String> list = new <>();
D) List<String> list = new ArrayList();
```

19. **Which class does not allow duplicate elements?** A) `ArrayList`
B) `HashSet`

C) `LinkedList`
D) `Vector`

20. **Which method is used to sort elements of a collection using a custom comparator?**

```
A) Collections.sort(list, comparator);
B) Collections.order(list, comparator);
C) Arrays.sort(list, comparator);
D) List.sort(list, comparator);
```

21. **Which interface is not part of the `.util.function` package?** A) `Predicate`
B) `Consumer`
C) `Function`
D) `Runnable`
22. **Which method of the `Stream` interface performs a reduction on the elements?** A) `map`
B) `filter`
C) `reduce`
D) `forEach`
23. **What does the `findFirst` method return when used on a stream?** A) The first element of the stream
B) An `Optional` describing the first element
C) The last element of the stream
D) A boolean indicating if the first element exists

Working with Stacks

24. **Which method in the stack checks if the stack is empty?**

```
A) isFull()
B) isEmpty()
C) size()
D) peek()
```

25. **What does the `push` method do in a stack implementation?**

```
A) Removes the top element
B) Adds an element to the top
C) Returns the top element without removing it
D) Checks if the stack is empty
```

Working with Queues

26. **Which method adds an element to the queue?**

```
A) enqueue()
B) dequeue()
C) isFull()
D) isEmpty()
```

27. **What does the `dequeue` method do in a queue implementation?**

```
A) Adds an element to the queue
B) Removes an element from the front
C) Checks if the queue is full
D) Returns the element at the front without removing it
```

Advanced Topics

28. **How do you create an unmodifiable list in ?**

```
A) List<Integer> list = Collections.unmodifiableList(new
ArrayList<>(Arrays.asList(1, 2, 3)));
B) List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3));
C) List<Integer> list = new UnmodifiableList<>(new
ArrayList<>(Arrays.asList(1, 2, 3)));
D) List<Integer> list = Arrays.asList(1, 2, 3);
```

29. **How do you iterate over a collection using a for-each loop in ?**

```
A) for (Element e : collection) {}
B) for (collection : Element e) {}
C) forEach (Element e in collection) {}
D) forEach (collection : Element e) {}
```

Practical Implementation Questions

30. **What does the following lambda expression do?**

```
(String s) -> s.toUpperCase()
```

A) Converts the input string to uppercase
B) Returns the length of the string
C) Checks if the string is empty
D) Returns the string unchanged

31. **Which `Stream` operation returns a list of unique elements?**

```
A) stream.distinct().collect(Collectors.toList())
B) stream.filter().collect(Collectors.toList())
```

```
C) stream.map().collect(Collectors.toList())
D) stream.sorted().collect(Collectors.toList())
```

32. **How do you create a `LocalDateTime` instance for a specific date and time?**

```
A) LocalDateTime.of(2024, Month.JULY, 25, 14, 30);
B) LocalDateTime.of(2024, 7, 25);
C) LocalDateTime.now();
D) LocalDateTime.of(2024, Month.JULY, 25);
```

33. **What is the output of the following code?**

```
List<String> list = Arrays.asList("a", "b", "c");
list.forEach(System.out::println);
```

A) `a b c`
B) `abc`
C) `c b a`
D) `a\nb\nc`

Concepts and Definitions

34. **Which design pattern ensures a class has only one instance and provides a global point of access to it?**

A) Factory
B) Singleton
C) Builder
D) Proxy

35. **What is a functional interface?**

A) An interface with more than one abstract method
B) An interface with exactly one abstract method
C) An interface with only static methods
D) An interface without any methods

Application and Analysis

36. **What is the output of the following lambda expression?**

```
Arrays.asList(1, 2, 3).stream().map(x -> x *
2).collect(Collectors.toList());
```

A) `[1, 2, 3]`
B) `[2, 4, 6]`

C) `[2, 3, 4]`
D) `[3, 4, 5]`

37. **Which method is used to format a `LocalDate`?**

```
A) date.format(DateTimeFormatter.ofPattern("yyyy-MM-dd"))
B) date.toString("yyyy-MM-dd")
C) date.format("yyyy-MM-dd")
D) date.pattern("yyyy-MM-dd")
```

Advanced 8 Features

38. **How do you convert a list of strings to a list of their lengths using streams?**

```
A) list.stream().map(String::length).collect(Collectors.toList())
B)
list.stream().mapToInt(String::length).collect(Collectors.toList())
C) list.stream().map(String::size).collect(Collectors.toList())
D) list.stream().mapToInt(String::size).collect(Collectors.toList())
```

39. **Which method would you use to handle an optional value?**

```
A) ifPresent()
B) get()
C) isEmpty()
D) isPresent()
```

Coding and Syntax

40. **What does the following code do?**

```
Stream.of("apple", "banana", "cherry")
      .filter(s -> s.startsWith("a"))
      .forEach(System.out::println);
```

A) Prints all elements of the stream
B) Prints `apple`
C) Prints `apple banana`
D) Prints `apple cherry`

Advanced Topics

41. **Which method of `CompletableFuture` is used to wait for the completion of all futures?**

```
A) CompletableFuture.allOf(futures)
B) CompletableFuture.anyOf(futures)
C) CompletableFuture.join(futures)
D) CompletableFuture.complete(futures)
```

42. **What is the output of the following code snippet?**

```
Optional<String> opt = Optional.of("Hello");
opt.ifPresent(System.out::println);
```

A) No output
B) `Optional[Hello]`
C) `Hello`
D) Exception

Collections and Data Structures

43. **Which method in `HashMap` retrieves a value based on a key?**

```
A) get()
B) put()
C) keySet()
D) values()
```

44. **What is the default initial capacity of `ArrayList`?** A) 8
B) 10
C) 16
D) 32

Practical Implementation

45. **How do you sort a list of integers in descending order using streams?**

```
A)
list.stream().sorted(Comparator.reverseOrder()).collect(Collectors.to
List())
B)
list.stream().sorted(Comparator.naturalOrder()).collect(Collectors.to
List())
C) list.stream().sorted().collect(Collectors.toList())
D)
list.stream().sorted(Comparator::reverseOrder).collect(Collectors.toL
ist())
```

46. **How do you check if a list contains a specific element using streams?**

```
A) list.stream().anyMatch(e -> e.equals(element))
B) list.stream().allMatch(e -> e.equals(element))
C) list.stream().noneMatch(e -> e.equals(element))
D) list.stream().filter(e -> e.equals(element))
```

Advanced Analysis

47. **What is the result of the following code?**

```
LocalDate date = LocalDate.parse("2024-07-25");
date.plusDays(10);
System.out.println(date);
```

A) `2024-07-25`
B) `2024-07-35`
C) `2024-08-04`
D) `Compilation error`

Deep Dive into Code

48. **Which of the following accurately describes a `Stack`?**

A) Last-In-First-Out (LIFO)
B) First-In-First-Out (FIFO)
C) Both A and B
D) None of the above

49. **Which method returns the number of elements in a collection?**

```
A) count()
B) size()
C) length()
D) getCount()
```

50. **How do you implement a thread-safe singleton in ?**

```
A) public class Singleton {
       private static Singleton instance;
       private Singleton() {}
       public static synchronized Singleton getInstance() {
           if (instance == null) {
               instance = new Singleton();
           }
           return instance;
       }
   }
B) public class Singleton {
       private static Singleton instance;
       private Singleton() {}
```

```
        public static Singleton getInstance() {
            if (instance == null) {
                instance = new Singleton();
            }
            return instance;
        }
    }
C) public class Singleton {
        private static volatile Singleton instance;
        private Singleton() {}
        public static Singleton getInstance() {
            if (instance == null) {
                synchronized (Singleton.class) {
                    if (instance == null) {
                        instance = new Singleton();
                    }
                }
            }
            return instance;
```