4

Q1. Write a program to show Interface Example in Java?

```
// Interface definition
interface MyInterface {
   void myMethod(); // Abstract method
}

// Class implementing the interface
class MyClass implements MyInterface {
   public void myMethod() {
      System.out.println("Implementation of myMethod() in MyClass");
   }
}

// Main class
public class InterfaceExample {
   public static void main(String[] args) {
      // Create an object of MyClass
      MyClass obj = new MyClass();

      // Call the method from the interface
      obj.myMethod();
   }
}
```

Q2. Write a program with 2 concrete methods and 2 abstract methods in Java?

```
// Abstract class
abstract class MyAbstractClass {
   // Concrete method
   public void concreteMethod1() {
      System.out.println("Concrete Method 1");
   }

   // Concrete method
   public void concreteMethod2() {
      System.out.println("Concrete Method 2");
   }

   // Abstract method
   public abstract void abstractMethod1();

   // Abstract method
   public abstract void abstractMethod2();
}

// Class extending the abstract class
class MyClass extends MyAbstractClass {
   // Implementation of abstractMethod1
   public void abstractMethod1() {
      System.out.println("Implementation of Abstract Method 1");
```

```
    }

    // Implementation of abstractMethod2
    public void abstractMethod2() {
        System.out.println("Implementation of Abstract Method 2");
    }
}

// Main class
public class AbstractExample {
    public static void main(String[] args) {
        // Create an object of MyClass
        MyClass obj = new MyClass();

        // Call the methods
        obj.concreteMethod1();
        obj.concreteMethod2();
        obj.abstractMethod1();
        obj.abstractMethod2();
    }
}
```

Q3. Write a program to show the use of functional interface in Java?

```
// Functional Interface
interface MyFunctionalInterface {
    void myMethod();
}

// Main class
public class FunctionalInterfaceExample {
    public static void main(String[] args) {
        // Using Lambda Expression to implement the functional interface
        MyFunctionalInterface obj = () -> {
            System.out.println("Implementation of myMethod() using Lambda Expression");
        };

        // Call the method
        obj.myMethod();
    }
}
```

Q4. What is an interface in Java?

In Java, an interface is a reference type that provides a contract or a set of abstract methods that a class must implement. It defines the methods that a class should have without specifying the implementation details.
An interface can contain abstract methods, default methods, and static methods. It can also define constants, which are implicitly public, static, and final. Interfaces are used to achieve abstraction and provide a way to achieve multiple inheritance in Java.

Q5. What is the use of an interface in Java?
The use of interfaces in Java includes:

1. Achieving Abstraction: Interfaces allow you to define a contract or a set of abstract methods that classes must implement. They provide a way to achieve abstraction by separating the definition of behavior from the

implementation details.

2. Implementing Multiple Inheritance: Unlike classes, Java supports multiple inheritance of interfaces. A class can implement multiple interfaces, allowing it to inherit behavior from multiple sources.
3. Enforcing Standards: Interfaces provide a way to define standards or specifications that classes must adhere to. By implementing an interface, a class guarantees that it will provide the specified behavior.
4. Promoting Code Reusability: Interfaces enable code reusability by defining a common set of methods that can be implemented by different classes. This allows different classes to share and reuse the same interface implementation.

Q6. What is the lambda expression in Java 8?
A lambda expression in Java 8 is a concise way to represent an anonymous function or a method without a name. It allows you to write code that can be treated as data and passed around as a parameter or assigned to a variable.

Q7. Can you pass lambda expressions to a method? When?
Yes, lambda expressions can be passed as arguments to methods. This is possible when the method parameter type is a functional interface, which is an interface with a single abstract method.

Q8. What is a functional interface in Java 8?
A functional interface in Java 8 is an interface that contains exactly one abstract method. It is also known as a SAM (Single Abstract Method) interface or a lambda-compatible interface.
Functional interfaces are designed to be used with lambda expressions and method references. They provide a way to represent behaviors or actions as data. Java 8 introduced the @FunctionalInterface annotation to indicate that an interface is intended to be a functional interface.

Q9. What is the benefit of lambda expressions in Java 8?

Lambda expressions in Java 8 provide several benefits, including:

5. Concise Syntax: Lambda expressions allow you to write more concise and readable code by eliminating the need for verbose anonymous inner classes.
6. Simplified Functional Programming: Lambda expressions make it easier to work with functional programming concepts like higher-order functions, closures, and functional interfaces.
7. Enhanced Code Reusability: Lambda expressions promote code reusability by allowing behavior to be treated as data and passed around as parameters or assigned to variables.
8. Improved Performance: Lambda expressions can enable more efficient execution of code by facilitating parallel processing and reducing the need for unnecessary intermediate objects.

Q10. Is it mandatory for a lambda expression to have parameters?
No, it is not mandatory for a lambda expression to have parameters. A lambda expression can have zero or more parameters, depending on the requirements of the functional interface it is implementing.
For eg:
```
list.forEach(()->System.out.println("hello"));
```