Assignment Questions 4 (java)

Q1.1. Write a program to show Interface Example in java?

Ans:

```
// Interface definition
interface Vehicle {
    void start();
    void stop();
}

// Class implementing the interface
class Car implements Vehicle {
    public void start() {
        System.out.println("Car started");
    }

    public void stop() {
        System.out.println("Car stopped");
    }
}

// Main program
public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        car.start();
        car.stop();
    }
}
```

Q2.Write a program a Program with 2 concrete method and 2 abstract method in java ?

Ans:

```
abstract class Shape {
    abstract void draw(); // Abstract method

    abstract void calculateArea(); // Abstract method

    void display() { // Concrete method
        System.out.println("Displaying shape.");
    }
```

```
    void getInfo() { // Concrete method
        System.out.println("Getting information about shape.");
    }
}

class Circle extends Shape {
    void draw() {
        System.out.println("Drawing a circle.");
    }

    void calculateArea() {
        System.out.println("Calculating area of a circle.");
    }
}

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle();
        circle.display();
        circle.getInfo();
        circle.draw();
        circle.calculateArea();
    }
}
```

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

Ans:

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

public class Main {
    public static void main(String[] args) {
        Calculator addition = (a, b) -> a + b;
        int result = addition.calculate(5, 3);
        System.out.println("Result of addition: " + result);

        Calculator subtraction = (a, b) -> a - b;
        result = subtraction.calculate(10, 4);
        System.out.println("Result of subtraction: " + result);
```

```
    }
}
```

Q4.What is an interface in Java?
Ans:In Java, an interface is a reference type that defines a contract for a class to follow. It specifies a set of methods that a class implementing the interface must provide. An interface cannot be instantiated directly but can be implemented by multiple classes. It enables the concept of multiple inheritance by allowing a class to implement multiple interfaces. Interfaces provide a way to achieve abstraction and define a common behavior that can be shared across different classes in a flexible and modular manner.

Q5.What is the use of interface in Java?

Ans: Interfaces in Java serve multiple purposes. They provide a way to achieve abstraction by defining a common set of methods that classes can implement. This allows for code reusability, as different classes can implement the same interface. Interfaces also enable loose coupling between classes, facilitating easier maintenance and extensibility. They play a vital role in achieving polymorphism, as objects of different classes implementing the same interface can be treated interchangeably. Interfaces are widely used in Java for creating contracts, defining API specifications, and supporting modular and flexible programming paradigms.

Q6.What is the lambda expression of Java 8?

Ans: In Java 8, lambda expressions were introduced as a concise way to represent anonymous functions. A lambda expression is a block of code that can be treated as a function and passed around as a parameter. It allows for more expressive and functional programming style. The syntax for a lambda expression is `(parameters) -> { body }`, where parameters represent the input arguments and the body contains the code to be executed. Lambda expressions enable the use of functional interfaces and provide a simpler and more readable way to implement them.

Q7.Can you pass lambda expressions to a method? When?

Ans:.Yes, lambda expressions can be passed as arguments to methods. This is possible when the method parameter type is a functional interface that matches the signature of the lambda expression. The lambda expression can be used as a concise way to provide the implementation of the functional interface's abstract method directly at the call site. This allows for flexible and dynamic behavior, where different implementations can be passed to the method, enabling customization and abstraction of functionality based on the specific requirements at runtime.

Q8.What is the functional interface in Java 8?
Ans: In Java 8, a functional interface is an interface that contains exactly one abstract method. It serves as the foundation for lambda expressions and method references. Functional interfaces are used to represent single-unit behaviors or actions. They enable functional programming in Java by providing a

way to treat functions as first-class citizens. The `@FunctionalInterface` annotation is used to indicate that an interface is intended to be functional. Functional interfaces can be used in lambda expressions, method references, and as targets for functional-style programming constructs such as streams and functional interfaces.

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

Ans: Lambda expressions in Java 8 provide several benefits. They enable concise and expressive code by allowing the representation of behavior as a block of code. Lambda expressions facilitate functional programming paradigms, allowing for easier composition and manipulation of functions. They enhance readability and maintainability by reducing boilerplate code. Lambda expressions also promote modular and reusable code, as they can be passed as arguments to methods and provide flexible and dynamic behavior. Additionally, lambda expressions enable better utilization of multi-core processors through parallel programming constructs like streams.

Q10.Is it mandatory for a lambda expression to have parameters?

Ans: No, it is not mandatory for a lambda expression to have parameters. Lambda expressions can be parameterless if the functional interface they are implementing does not require any input parameters. In such cases, the parameter list of the lambda expression can be empty, denoted by empty parentheses. For example, `() -> { // body }`. This allows for the representation of functions or actions that do not require any input arguments, providing flexibility and conciseness in code implementation.