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

Ans

```
interface Shape {
  void draw();
}

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

class Square implements Shape {
  public void draw() {
    System.out.println("Drawing a square");
  }
}

public class Main {
  public static void main(String[] args) {
    Shape circle = new Circle();
    Shape square = new Square();

    circle.draw();
    square.draw();
  }
}
```

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

Ans

```
abstract class Shape {
  public void draw() {
    System.out.println("Drawing a shape");
  }
  public void wish() {
    System.out.println("Good luck");
  }

  public abstract void calculateArea();

  public abstract void calculatePerimeter();
}

class Circle extends Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }
```

```
  @Override
  public void calculateArea() {
    System.out.println("The area of the circle is: " + Math.PI * radius * radius);
  }

  @Override
  public void calculatePerimeter() {
    System.out.println("The perimeter of the circle is: " + 2 * Math.PI * radius);
  }
}

class Square extends Shape {
  private double side;

  public Square(double side) {
    this.side = side;
  }

  @Override
  public void calculateArea() {
    System.out.println("The area of the square is: " + side * side);
  }

  @Override
  public void calculatePerimeter() {
    System.out.println("The perimeter of the square is: " + 4 * side);
  }
}

public class Main {
  public static void main(String[] args) {
    Shape circle = new Circle(5);
    Shape square = new Square(4);

    circle.draw();
    circle.calculateArea();
    circle.calculatePerimeter();
    circle.wish();

    square.draw();
    square.calculateArea();
    square.calculatePerimeter();
    square.wish();
  }
}
```

Ans

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

Ans

```
import java.util.function.Function;
```

```
public class Main {
  public static void main(String[] args) {
    Function<String, Integer> lengthOfString = String::length;

    System.out.println("The length of the string \"Hello\" is: " + lengthOfString.apply("Hello"));
  }
}
```

Sure, here is a program to show the use of functional interface in Java:

Java

```
import java.util.function.Function;

public class Main {
  public static void main(String[] args) {
    Function<String, Integer> lengthOfString = String::length;

    System.out.println("The length of the string \"Hello\" is: " + lengthOfString.apply("Hello"));
  }
}
```

This program defines a function interface called Function. The Function interface has a single abstract me thod called apply(). The apply() method takes one argument and returns a result.

The main() method creates a function object that implements the Function interface. The function object u ses the String::length method to get the length of a string.

The main() method then calls the apply() method on the function object. The apply() method takes the stri ng "Hello" as an argument and returns its length.

Q4.What is an interface in Java?

Ans

An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a behaviour. A Java interface contains static constants and a bstract methods.

Q5.What is the use of interface in Java?

Ans

Interfaces are used in Java to achieve abstraction, multiple inheritance, and loose coupling.

Abstraction: An interface specifies the behavior of a class, but it does not specify how that behavior is imp lemented. This allows us to focus on the behavior of an object without worrying about how that behavior is implemented. This makes our code more reusable and maintainable.

Multiple inheritance: In Java, a class can only extend one class. However, by using interfaces, a class can implement multiple interfaces. This allows us to reuse code and create more flexible and adaptable class es.

Loose coupling: An interface defines a contract between two classes. This means that the two classes do not need to know about each other's implementation details. This makes our code more flexible and adaptable.

Q6.What is the lambda expression of Java 8?

Ans

A lambda expression in Java 8 is a short block of code that can be used to represent an anonymous function. It is a concise way to express the logic of a small function without having to define a separate class or method.

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

Ans

Yes, you can pass lambda expressions to a method. This is called method reference. A method reference is a way to refer to a method without having to explicitly invoke it.

You can pass a lambda expression to a method if the method's parameter is a functional interface. A functional interface is an interface that has only one abstract method.

Q8.What is the functional interface in Java 8?

Ans

In Java 8, a functional interface is an interface that has only one abstract method. This makes it possible to use lambda expressions to implement the interface. Lambda expressions are short blocks of code that can be used to represent anonymous functions. Using lambda expressions with functional interfaces can make code more concise and reusable.

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

Ans

Lambda expressions in Java 8 offer a number of benefits, including:

Conciseness: Lambda expressions can make code more concise and readable. This is because they can be used to represent anonymous functions without having to define a separate class or method.

Reusability: Lambda expressions can be reused in different places in your code. This can make your code more modular and easier to maintain.

Functional programming: Lambda expressions can be used to write code in a functional programming style. Functional programming is a programming paradigm that focuses on the use of functions. This can make your code more concise, readable, and reusable.

Stream operations: Lambda expressions can be used to perform stream operations. Stream operations are a way to process collections of data in a functional way. This can make your code more concise, readable, and efficient.

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

Ans

No, it is not mandatory for a lambda expression to have parameters. A lambda expression can have zero, one, or multiple parameters.