Class Notes 12: Interface

In Java, an interface is a blueprint for a class, similar to an abstract class but with some key differences. An interface defines a set of abstract methods that any class implementing the interface must provide concrete implementations for. It provides a way to achieve multiple inheritance in Java because a class can implement multiple interfaces.

Here's the basic syntax for declaring an interface in Java:

public interface MyInterface { // Declare abstract methods (method signatures) here

void method1();

int method2(String str);

}

Now, let's break down the key elements of interfaces:

1. **interface Keyword**: You declare an interface using the **interface** keyword.
2. **Method Signatures**: Inside the interface, you define method signatures without providing the method bodies. These are abstract methods, meaning they don't have any code associated with them.
3. **Implementing an Interface**: Any class that wants to use an interface must implement it using the **implements** keyword.

How to use interfaces in java

```
interface FirstInterface {
  public void myMethod(); // interface method
}


interface SecondInterface {
  public void myOtherMethod(); // interface method
}
class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
```

```
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}
class Main {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
  }
}
```

Here's another example:

```
interface Shape {
double area();
double perimeter();
} // Implement the interface in a class
class Circle implements Shape {
private double radius;
public Circle(double radius)
{
 this.radius = radius;
 }
@Override public double area()
{
return Math.PI * radius * radius;
```

```
}
@Override public double perimeter() {
return 2 * Math.PI * radius;
}
 }
class Rectangle implements Shape {
private double length;
private double width;
public Rectangle(double length, double width)
 {
   this.length = length;
   this.width = width;
}
@Override public double area() {
 return length * width;
}
@Override public double perimeter()
{
return 2 * (length + width);
}
}
public class Main {
 public static void main(String[] args) {
Circle circle = new Circle(5.0);
Rectangle rectangle = new Rectangle(4.0, 3.0);
System.out.println("Circle Area: " + circle.area());
System.out.println("Circle Perimeter: " + circle.perimeter());
System.out.println("Rectangle Area: " + rectangle.area());
```

```
System.out.println("Rectangle Perimeter: " + rectangle.perimeter());

}

}
```

In this example, we define an interface **Shape** with two abstract methods: **area()** and **perimeter()**. We then create two classes, **Circle** and **Rectangle**, both of which implement the **Shape** interface. These classes provide concrete implementations for the **area()** and **perimeter()** methods.

Interfaces are commonly used to define contracts that classes must adhere to, ensuring a consistent API for various implementations. They're particularly useful when you want to define a common set of methods for classes that may not share the same inheritance hierarchy.