Abstract Classes and Interfaces in Java

Introduction

In Java, both **abstract classes** and **interfaces** are used to achieve abstraction, which allows developers to define methods that must be implemented by subclasses or implementing classes. Although they share similarities, they serve different purposes and have distinct characteristics.

Abstract Classes

An abstract class in Java is a class that cannot be instantiated. It can contain both abstract methods (without implementations) and concrete methods (with implementations). Abstract classes are defined using the `abstract` keyword.

Characteristics of Abstract Classes

- Can have both abstract and non-abstract methods.
- Can have constructors.
- Can have instance variables.
- A subclass can extend only one abstract class (single inheritance).

Example of an Abstract Class

```
1 // Abstract class
2 abstract class Animal {
3     String name;
```

```
4
5       // Constructor
6       Animal(String name) {
7           this.name = name;
8       }
9
10      // Abstract method
11      abstract void makeSound();
12
13      // Concrete method
14      void eat() {
15          System.out.println(name + " is eating.");
16      }
17  }
18
19  // Subclass
20  class Dog extends Animal {
21      Dog(String name) {
22          super(name);
23      }
24
25      @Override
26      void makeSound() {
27          System.out.println("Woof! Woof!");
28      }
29  }
```

Interfaces

An interface in Java is a reference type that contains abstract methods and static constants. Interfaces are implemented by classes using the `implements` keyword.

Characteristics of Interfaces

- All methods are implicitly public and abstract (except default and static methods).
- Cannot contain constructors.

- Can have static and default methods (from Java 8 onwards).
- A class can implement multiple interfaces (multiple inheritance).

Example of an Interface

```java
// Interface
interface Vehicle {
    int getNumberOfWheels();
    void drive();
}

// Implementing class
class Car implements Vehicle {
    @Override
    public int getNumberOfWheels() {
        return 4;
    }

    @Override
    public void drive() {
        System.out.println("Driving a car...");
    }
}

// Another implementing class
class Bicycle implements Vehicle {
    @Override
    public int getNumberOfWheels() {
        return 2;
    }

    @Override
    public void drive() {
        System.out.println("Riding a bicycle...");
    }
}
```

Differences Between Abstract Classes and Interfaces

Feature	Abstract Class	Interface
Method Types	Abstract and Concrete	Abstract, Default, and Static
Variables	Instance, Final, Static	Static and Final only
Inheritance	Single inheritance	Multiple inheritance
Constructors	Allowed	Not allowed

Exercises

1. **Abstract Class Exercise:** Create an abstract class `Shape` with two abstract methods: `calculateArea()` and `calculatePerimeter()`. Implement concrete subclasses `Circle` and `Rectangle` to provide specific implementations for these methods. Test your code with different inputs.

2. **Interface Exercise:** Design an interface `Playable` with a method `play()`. Create classes `Piano` and `Guitar` that implement the interface. Write a program that uses a list of `Playable` objects to invoke the `play()` method.

3. **Comparison Exercise:** Write a Java program that uses both an abstract class and an interface. Create an abstract class `Appliance` with a method `turnOn()`, and an interface `EnergyEfficient` with a method `calculateEfficiency()`. Implement these in a class `WashingMachine`.

4. **Interface with Default Methods Exercise:** Create an interface `Logger` with a default method `log(String message)` that prints a message with a timestamp. Create a class `FileLogger` that implements `Logger` and overrides the `log` method to write messages to a file instead.