Lab 28

Java Abstraction

Abstraction is one of the core principles of object-oriented programming (OOP) and is used to manage the complexity of software systems by hiding unnecessary details while exposing the essential features. In Java, abstraction is achieved through abstract classes and interfaces.

Abstraction:

Abstraction is the process of simplifying complex reality by modeling classes based on the essential properties and behaviors they should have, while ignoring the non-essential details. It allows you to create a blueprint for a class without providing a complete implementation. Abstraction is about defining a clear and concise interface for interacting with objects.

Abstract Class:

Abstract classes are classes that cannot be instantiated on their own and are designed to be subclassed. They can contain both abstract (unimplemented) methods and concrete (implemented) methods. Abstract methods are declared without providing an implementation in the abstract class, leaving it to the subclasses to provide the implementation.

Key points about abstract classes:

1. Abstract classes are defined using the `abstract` keyword.

2. They can have both abstract and concrete methods.

3. You cannot create an instance of an abstract class using the `new` keyword.

4. Subclasses of an abstract class must provide concrete implementations for all abstract methods, unless the subclass itself is also declared as abstract.

Here's an example of an abstract class:

abstract class Shape {

// Abstract method (no implementation)

abstract double area();

// Concrete method

void display() {

System.out.println("This is a shape.");

}

}

class Circle extends Shape {

double radius;

Circle(double r) {

radius = r;

}

@Override

double area() {

return Math.PI \* radius \* radius;

}

}

public class AbstractionDemo {

public static void main(String[] args) {

Circle circle = new Circle(5.0);

System.out.println("Area of circle: " + circle.area());

circle.display();

}

}

In this example, `Shape` is an abstract class with an abstract method `area()`. The `Circle` class extends `Shape` and provides an implementation for the `area()` method. Instances of `Circle` can be created and used.

Interface:

An interface in Java is a collection of abstract methods (methods without a body) and constants (final variables). It defines a contract that implementing classes must adhere to. Interfaces allow for multiple inheritance in Java, as a class can implement multiple interfaces.

Key points about interfaces:

1. Interfaces are defined using the `interface` keyword.

2. All methods in an interface are implicitly `public` and `abstract` (no need to use `abstract` and `public` keywords).

3. Interface variables are implicitly `public`, `static`, and `final` (constants).

4. A class that implements an interface must provide concrete implementations for all the methods declared in the interface.

Here's an example of an interface:

interface Animal {

void makeSound();

}

class Dog implements Animal {

@Override

public void makeSound() {

System.out.println("Dog barks.");

}

}

class Cat implements Animal {

@Override

public void makeSound() {

System.out.println("Cat meows.");

}

}

public class InterfaceDemo {

public static void main(String[] args) {

Animal dog = new Dog();

Animal cat = new Cat();

dog.makeSound();

cat.makeSound();

}

}

In this example, the `Animal` interface defines an abstract method `makeSound()`, which is implemented by the `Dog` and `Cat` classes. Instances of `Dog` and `Cat` can be treated as `Animal` objects, demonstrating the concept of polymorphism and interface-based abstraction.

In summary, abstraction, abstract classes, and interfaces are important concepts in Java that allow you to create clear and modular code by defining a contract for classes to follow. Abstraction helps manage complexity and promotes code reusability and flexibility in object-oriented programming.