Software Design Patterns CCSW 413 Lab Manual

Lab 1: Introduction to Advanced Java Concepts

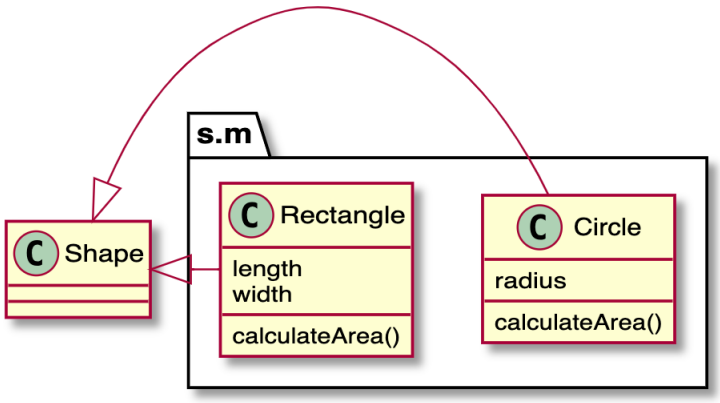
Objectives

1. Gain understanding of interfaces in Java.
2. Gain understanding of inheritance and polymorphism.
3. Learn the concept of abstract classes and their usage.
4. Differentiate clearly between overloading and overriding.
5. Comprehend the concepts of object declared type and concrete type.

Lab Content

1. Interfaces

Definition: An interface in Java is a collection of abstract methods. It defines a contract for classes to implement, allowing multiple inheritance and providing a way for classes to be related without the need for a common base class.




```java
interface Shape {
    double calculateArea();
}

class Circle implements Shape {
    private double radius;

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

    @Override
    public double calculateArea() {
```

```
        return Math.PI * radius * radius;
    }
}

class Rectangle implements Shape {
    private double length;
    private width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    public double calculateArea() {
        return length * width;
    }
}
```

Extended Example: Consider a shape hierarchy with a base class `Shape`:

```
class Shape {
    void draw() {
        System.out.println("Drawing␣a␣shape");
    }
}
```

Now, create specific shapes like `Circle` and `Square` that extend `Shape`:

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

class Square extends Shape {
    void draw() {
        System.out.println("Drawing␣a␣square");
    }
}
```

Now, polymorphism allows using these shapes interchangeably:

```
Shape myShape = new Circle();
myShape.draw(); // Calls the draw method in Circle
```

2. Inheritance and Polymorphism

Definition: Inheritance in Java is a mechanism where a new class inherits properties and behaviors from an existing class. Polymorphism allows objects of different types to be treated as objects of a common type.

```
class Animal {
    void makeSound() {
        System.out.println("Animal␣makes␣a␣sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog␣barks");
```

```
    }
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println("Cat meows");
    }
}
```

Elaboration: Inheritance is a way to achieve code reusability. In the example, `Dog` and `Cat` inherit the `makeSound()` method from `Animal`. Polymorphism allows us to treat `Dog` and `Cat` objects as `Animal` objects, enabling dynamic method invocation.

3. Abstract Classes

Definition: An abstract class in Java is a class that cannot be instantiated and may contain abstract methods. It allows the definition of common methods that subclasses must implement.

```
abstract class Shape {
    abstract double calculateArea();
}

class Circle extends Shape {
    private double radius;

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

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```

4. Overloading and Overriding

Definition: Overloading is the ability to define multiple methods with the same name in a class, but with different parameters. Overriding is the ability of a subclass to provide a specific implementation of a method that is already defined in its superclass.

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}
```

Elaboration: Overloading allows a class to have multiple methods with the same name but different parameters. For example, the `add()` method in `Calculator` is overloaded based on the parameter types.

5. Object Declared Type and Concrete Type

Definition: The declared type of an object is the type declared at compile time, while the concrete type is the actual type of the object at runtime.

```
Animal myDog = new Dog(); // Declared type is Animal, concrete type is Dog
```

Real-life Example: Imagine a library system where there's a general `LibraryItem` class:

```
class LibraryItem {
    void checkout() {
        System.out.println("Item checked out");
    }
}
```

Lab Exercises

Exercise 1: Implementing Interfaces

Task: Create an interface `Drawable` with a method `draw()`. Implement it in two classes: `Circle` and `Square`. Create instances of both classes and invoke the `draw()` method.

Instructions: After implementing the solution, draw a class diagram representing the classes `Drawable`, `Circle`, and `Square`. Include associations and any relevant attributes or methods.

Exercise 2: Inheritance and Polymorphism

Task: Create a base class `Vehicle` with a method `start()` and two subclasses: `Car` and `Bicycle`. Override the `start()` method in both subclasses. Create instances of both subclasses, store them in an array of `Vehicle`, and invoke the `start()` method for each.

Instructions: After implementing the solution, draw a class diagram representing the classes `Vehicle`, `Car`, and `Bicycle`. Include associations and any relevant attributes or methods.

Exercise 3: Abstract Classes

Task: Create an abstract class `Employee` with a method `calculateSalary()`. Extend this class with two subclasses: `Manager` and `Developer`. Implement the `calculateSalary()` method in each subclass with different formulas. Create instances of both subclasses and calculate their salaries.

Instructions: After implementing the solution, draw a class diagram representing the classes `Employee`, `Manager`, and `Developer`. Include associations and any relevant attributes or methods.

Exercise 4: Overloading and Overriding

Task: Create a class `MathOperations` with overloaded methods for addition, subtraction, and multiplication. Extend this class to create a subclass `AdvancedMathOperations` and override the multiplication method. Demonstrate the use of both classes.

Instructions: After implementing the solution, draw a class diagram representing the classes `MathOperations` and `AdvancedMathOperations`. Include associations and any relevant attributes or methods.

Exercise 5: Object Declared Type and Concrete Type

Task: Create a class hierarchy for zoo animals with a base class `Animal` and subclasses `Lion`, `Elephant`, and `Monkey`. Use object declared types and concrete types to store instances in an array and invoke specific methods.

Instructions: After implementing the solution, draw a class diagram representing the classes `Animal`, `Lion`, `Elephant`, and `Monkey`. Include associations and any relevant attributes or methods.

Evaluation Rubric

- **Understanding (20%):** Students' answers to exercises demonstrate a clear understanding of interfaces, inheritance, polymorphism, abstract classes, overloading, overriding, and object declared type vs. concrete type.
- **Implementation (60%):** Successfully completes the lab exercises with a correct implementation of concepts.
- **Documentation (20%):** Provides clear class diagrams and the code is well-commented and easy to follow.