ALGONQUIN COLLEGE

CST8132 OBJECT ORIENTED PROGRAMMING

Properties of OOP

- Abstract class, Interface

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

Today's Topics

- Features of OOP
  - Abstract Classes
  - Interfaces
- Access Modifiers
- Type Casting
- Lab 3 – College System I

Abstract Class - Example

```java
public abstract class Shape {
    private String name;
    protected double area;
    protected double perimeter;

    Shape(){}

    Shape(String n){
        name = n;
    }

    public abstract void findArea();
    public  abstract void findPerimeter();


    public void printDetails() {
        System.out.println("Name: " + name + "  Area: " + area +
                " Perimeter: "+ perimeter);
    }

    public void printNum() {
        System.out.println("from Shape");
    }
}
```

```java
public class Circle extends Shape{

    private double radius;

    Circle( int r){
        super("Circle");
        //name = "Circle";
        radius = r;

    }

    @Override
    public void findArea() {
        area = 2*Math.PI * Math.pow(radius, 2);
    }

    @Override
    public void findPerimeter() {
        perimeter = 2*Math.PI * radius;
    }

    @Override
    public void printNum() {
        System.out.println("From Circle");
    }
}
```

```java
public class Rectangle extends Shape{

    private int length;
    private int width;

    Rectangle(){

    }

    Rectangle( int len, int wid){
        //name = "Rectangle";
        super("Rectangle");
        length = len;
        width = wid;
    }
    @Override
    public void findArea() {
        area = length * width;
    }
    @Override
    public void findPerimeter() {
        perimeter = 2*(length + width);
    }
}
```

```java
public class Square extends Shape{

    private double side;

    Square(int num){
        super("Square");
        //name = "Square";
        side = num;
    }
    @Override
    public void findArea() {
        area = side * side;
    }

    @Override
    public void findPerimeter() {
        perimeter = 4 * side;
    }
}
```

Comments NOT included to save space.

Abstract Class - Example

```
public class ShapeTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Rectangle r1 = new Rectangle(5,10);
        r1.findArea();
        r1.findPerimeter();
        r1.printDetails();

        Circle c1 = new Circle(3);
        c1.findArea();
        c1.findPerimeter();
        c1.printDetails();

//      Shape s1 = new Shape();
//      s1.findArea();

        Scanner input = new Scanner(System.in);
        System.out.println("Number of shapes you want to create: ");
        int num = input.nextInt();

        Shape []shapes = new Shape[num];  // shapes is an array that can store Shape objects in it

        for(int i=0; i<shapes.length; i++) {
            System.out.println("1. Square \n2. Rectangle \nEnter option");
            int option = input.nextInt();
            if(option == 1)
                shapes[i] = new Square(3);  // each object should be created. otherwise, nullPointerException
            else if (option == 2)
                shapes[i] = new Rectangle(4,5);

            shapes[i].findArea();
            shapes[i].findPerimeter();
        }

        for(int i=0; i<shapes.length; i++)
            shapes[i].printDetails();
    }
}
```

Abstract classes and methods

- Abstract classes cannot be instantiated
  - `public abstract class Shape{...`
- Abstract classes are used to form a basis for subclasses
  - E.g. Our Shape class is an abstract class
  - Provides information about how to deal with shapes of all kinds (e.g. can calculate area of shapes)
  - The “real” shapes like triangle and circle inherit from Shape (extends Shape)
  - The subclasses must provide method bodies for the abstract methods in the superclass
- Abstract methods are used to establish a behavior without writing the code to specify the details of how it should be carried out
  - Abstract methods are meant to be overridden (@Override)

Interfaces

- If an abstract class has nothing other than
  - Public static final properties
  - Abstract methods

  Then it could be made into an interface by changing

```
public abstract class Shape{
```

  into

```
public interface Shape{
```

  - This is because an interface is a list of abstract methods (and if it has any properties, then they are `public static final`)
    - In fact, as all methods in an interface are `abstract`, there is no need to declare them as `abstract`
    - Because all properties (if present) are `public static final`, there is no need to declare them as `public static final`

Interfaces

- An interface tells us what we can do with an object, and how we do it
- An interface provides us with a set of method signatures
  - Signature: method name, number and type of parameters
- An interface provides us the return values of methods
- In other words, an interface tells us how to use the object that implements the interface

Interface – Example

```java
public interface Shape {
    public static final String name = "Shape";

    public void findArea();
    public void findPerimeter();
    public void printDetails();
}
```

```java
public class Square implements Shape{

    private double side;
    protected String name;
    protected double area;
    protected double perimeter;

    @Override
    public void findArea() {
        area = side * side;
    }

    @Override
    public void findPerimeter() {
        perimeter = 4 * side;

    }

    @Override
    public void printDetails() {
        System.out.println("Name: " + name + "  Area: " + area
                + " Perimeter: "+ perimeter);
    }
}
```

```java
public class Circle implements Shape{

    private double radius;
    protected String name;
    protected double area;
    protected double perimeter;

    Circle (){}

    Circle( int r){
        name = "Circle";
        radius = r;
    }

    @Override
    public void findArea() {
        area = Math.pow(radius, 2);
    }

    @Override
    public void findPerimeter() {
        perimeter = 2*Math.PI * radius;
    }

    @Override
    public void printDetails() {
        System.out.println("Name: " + name + "  Area: " + area
                + " Perimeter: "+ perimeter);
    }

}
```

```java
public class Rectangle implements Shape{

    private int length;
    private int width;
    protected String name;
    protected double area;
    protected double perimeter;

    Rectangle( int len, int wid){
        name = "Rectangle";
        length = len;
        width = wid;
    }

    @Override
    public void findArea() {
        area = length * width;
    }

    @Override
    public void findPerimeter() {
        perimeter = 2*(length + width);
    }

    @Override
    public void printDetails() {
        System.out.println("Name: " + name + "  Area: "
                + area + " Perimeter: "+ perimeter);
    }

}
```

Comments NOT included to save space.

Interface – Example

```
public class ShapeTest {

    public static void main(String[] args) {

        Rectangle r1 = new Rectangle(5,10);
        r1.findArea();
        r1.findPerimeter();
        r1.printDetails();

        Circle c1 = new Circle(3);
        c1.findArea();
        c1.findPerimeter();
        c1.printDetails();

        //Shape s1 = new Shape();
    }
}
```

Multiple Inheritance

- In Java, a class can extend **at most** one super class – this is contrary to the notion of multiple inheritance
- In Java, interfaces provide an alternative to multiple inheritance
- A class can implement more than one interface
- Example:

```
public class Rock implements Hammer, PaperWeight, Weapon {

   }
```

- Public class Employee extends Person, Hr{… } // will not work
- Multilevel – Employee extends Person, Regular extends Employee // possible

`static` keyword

- Discussion of `static` versus instance
  - Static variables and static methods are a part of the class itself
  - All objects of the class, if any, use the SAME copy of them
  - Static variables and methods can be used BEFORE any objects of that class are instantiated
  - Static members should be referenced by the `ClassName` itself
  - Example of using static method “sort” of Arrays class:
    - `Arrays.sort(myArray)`

    (reference for Arrays class: https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html)
  - Instance members do not exist until an object has been created (with the **new** keyword)
  - Instance members are accessed through a reference to the object

Access Modifiers

- Private
  - Only instances of the class itself can access it
- Protected
  - Only instances of the class and instances of the subclasses can access
- Public
  - Everyone has access. can be accessed from within the class, outside the class, within the package and outside the package
- Default
  - Members of the same package can access all members

Access modifiers (contd.)

Access Modifier	Java keyword	UML symbol	Within class	Within package	Is subclass	Not subclass
Public	public	+	Y	Y	Y	Y
Protected	protected	#	Y	Y	Y	N
Package		~	Y	Y	N	N
Private	private	-	Y	N	N	N

Lab 3 – College System I