Object-Orientated Programming

CMM24

RGU (2023)

Session 9 (Object Orientated Programming)

Introduction to enums

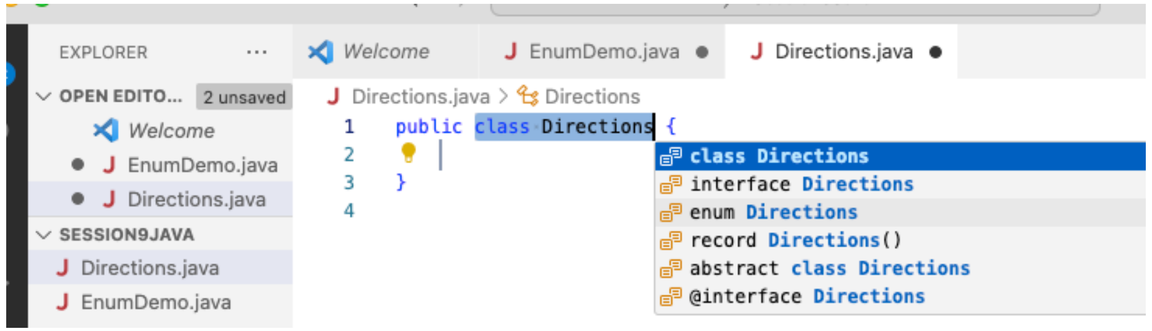
Introduction to abstract classes

CMM024

Summary

- **Discuss enum**
- **Discuss abstract classes**
- **Why using enums and abstract classes**
- **Go through an example using the Shapes concepts**
- **Testing the model**

What are enums?




```
public enum Directions {
    NORTH,
    SOUTH,
    EAST,
    WEST
}
```

```
public enum DaysOfWeek {
    MONDAY, TUESDAY, WEDNESDAY,THURSDAY,FRIDAY,SATURDAY,SUNDAY
}
```

- An enum type is a special data type that enables for a variable to be a set of predefined constants.
- The variable must be equal to one of the values that have been predefined for it.
- Example: Directions (values of NORTH, SOUTH, EAST, and WEST) or the days of the week (MONDAY, TUESDAY, WEDNESDAY,THURSDAY,FRIDAY,SATURDAY,SUNDAY).

How to use enums

```
public class Person {

    private String name;
    private Gender gender;

    public Person(String name, Gender gender ){
        this.name = name;
        this.gender = gender;
    }

    public Gender getGender(){
        return gender;
    }

    public String getName(){
        return name;
    }

    public String toString(){
        String st = "";
        st += "Name: " + this.name;
        st += "\tGender: " + this.gender.toString();
        return st;
    }
}
```

```
public class Child extends Person{

    public Child(String name, Gender gender){
        super(name, gender);
    }
}
```

```
public class Parent extends Person{
    public Parent(String name, Gender gender){
        super(name, gender);
    }
}
```

```
import java.util.ArrayList;

public class EnumDemo {

    public static void main(String[] args){

        Person p1 = new Child("David", Gender.MALE);
        Person p2 = new Child("Mary", Gender.FEMALE);
        Person p3 = new Parent("Robert", Gender.MALE);
        Person p4 = new Parent("Suzy", Gender.FEMALE);

        ArrayList<Person> people = new ArrayList<>();
        people.add(p1);
        people.add(p2);
        people.add(p3);
        people.add(p4);

        for(Person person: people){

            // if(person.getGender() == Gender.FEMALE){
            //     System.out.println(person);
            // }

            if(person.getGender() == Gender.FEMALE && person instanceof Parent){
                System.out.println("Parents who are females");
                System.out.println(person);
            }
        }

    }

}
```

What are abstract classes

- An abstract class is a class that is declared abstract
  - Abstract classes cannot be instantiated, but they can be subclassed.
  - They may have concrete methods (implemented)
- it may or may not include abstract methods.
  - An abstract method is a method that is declared without an implementation
- There are times when some functionality must be implemented but the superclass has not got all the information to provide that functionality.
  - The subclasses are the only ones who can provide this functionality
  - Creating abstract methods pushes the subclasses to have to implement the abstract method
  - The superclass can invoke the method and java will automatically ask the right subclass to run the method.

How using abstract classes: BasicShape

- Let’s try to create a class to model some basic shape.
- The shape has a type and a colour.
- Note that all subclasses will have to implement the getDetails method!

```
public enum Color {
    WHITE,
    BLACK,
    BLUE,
    RED,
    ORANGE,
    GREEN,
    YELLOW
}
```

```
public abstract class BasicShape {
    private Color color;

    public BasicShape(Color color) {
        this.color = color;
    }

    public Color getColor() {
        return this.color;
    }

    public abstract String getDetails();

    public String toString(){
        return getDetails();
    }
}
```

How using abstract classes: Shape2D

- Now there are two types of shapes
- 2D I.e. circles , rectangles etc
- 3D I.e, spheres, cubes etc
- We need to create another set of subclasses to model these.
- Let call them Shape2D and Shape3D
- Any shapes have a X and Y poisition on the canvas!
- They both Extends BasicShape

```
public abstract class Shape2D extends BasicShape{

    private int xPos;
    private int yPos;
    public Shape2D(Color color, int xPos, int yPos) {
        super(color);
        this.xPos = xPos;
        this.yPos = yPos;
    }
    public int getXPos() {
        return this.xPos;
    }
    public int getYPos() {
        return this.yPos;
    }
    public abstract double getPerimeter();
    public abstract double getArea();

    public String getDetails() {
        String st = "";
        st += "\nColor: " + this.getColor();
        st += "\nX position: " + this.getXPos();
        st += "\nY position: " + this.getYPos();
        st += "\nPerimeter: " + String.format("%.2f", this.getPerimeter());
        st += "\nArea: " + String.format("%.2f", this.getArea());
        return st;
    }
}
```

How using abstract classes: Shape3D

- Now there are two types of shapes
- 2D I.e. circles , rectangles etc
- 3D I.e, spheres, cubes etc
- We need to create another set of subclasses to model these.
- Let call them Shape2D and Shape3D
- Any shapes have a X and Y position for 2D and X,Y and Z for 3D on the canvas!
- They both Extends BasicShape

```
public abstract class Shape3D extends BasicShape {

    private int xPos;
    private int yPos;
    private int zPos;

    public Shape3D(olor color, int xPos, int yPos, int zPos) {
        super(shapeType, color);
        this.xPos = xPos;
        this.yPos = yPos;
        this.zPos = zPos;
    }
    public int getXPos() {
        return this.xPos;
    }
    public int getYPos() {
        return this.yPos;
    }
    public int getZPos() {
        return this.zPos;
    }
   public abstract double getVolume();
   public String getDetails() {
        String st = "";
        st += "\nColor: " + this.getColor();
        st += "\nX position: " + this.getXPos();
        st += "\nY position: " + this.getYPos();
        st += "\nY position: " + this.getZPos();
        st += "\nVolume: " + String.format("%.2f", this.getVolume());
        return st;
    }
}
```

Let create a real 2D shape: Rectangle

```
public class Rectangle extends Shape2D{
    private  double width;
    private double length;
    public Rectangle(Color color, int xPos, int yPos, double width, double length){
        super(color, xPos, yPos);
        this.width = width;
        this.length = length;
    }
    public double getWidth(){
        return this.width;
    }
    public double getLength(){
        return this.length;
    }
    @Override
    public double getPerimeter() {
       //P= 2 * π * r
       return (2 * this.width) + (2 * this.length);
    }
    @Override
    public double getArea() {
       //A = π * r ^ 2
       return width * length;
    }
}
```

Let create a real 2D shape: Rectangle

```
public class Rectangle extends Shape2D{
    private  double width;
    private double length;
    public Rectangle(Color color, int xPos, int yPos, double width, double length){
        super(color, xPos, yPos);
        this.width = width;
        this.length = length;
    }
    public double getWidth(){
        return this.width;
    }
    public double getLength(){
        return this.length;
    }
    @Override
    public double getPerimeter() {
       //P= 2 * π * r
       return (2 * this.width) + (2 * this.length);
    }
    @Override
    public double getArea() {
       //A = π * r ^ 2
       return width * length;
    }
}
```

- Rectangle extends Shape2D
- A rectangle has a width and length
- Both abstract methods from Shape2D are implemented

Let create a real 2D shape: Circle

```
public class Circle extends Shape2D{
    private  double radius;
    public Circle(ShapeType shapeType, Color color, int xPos, int yPos, double radius){
        super(shapeType, color, xPos, yPos);
        this.radius = radius;
    }
    public double getRadius(){
        return this.radius;
    }
    @Override
    public double getPerimeter() {
       //P= 2 * π * r
       return 2 * Math.PI * this.radius;
    }
    @Override
    public double getArea() {
       //A = π * r ^ 2
       return Math.PI * this.radius * this.radius;
    }
}
```

- Circle al extends Shape2D
- A circle has a radius only
- Both abstract methods from Shape2D are implemented

Let create a real 3D shape: Sphere

```
public class Sphere extends Shape3D{

    private double radius;

    public Sphere(ShapeType shapeType, Color color, int xPos, int yPos, int zPos, double radius){
        super(shapeType, color, xPos, yPos, zPos);
        this.radius = radius;
    }

    public double getRadius(){
        return this.radius;
    }

    @Override
    public double getVolume() {
        return (4.0/3.0) * Math.PI * Math.pow(radius, 3);
    }
}
```

- Sphere extends Shape2D
- A sphere is define by its radius
- The abstract method from Shape3D is implemented

How using abstract classes

```
import java.util.ArrayList;

public class ShapesTester {

    public static void main(String[] args) {
        BasicShape s1 = new Circle(Color.BLUE, 10, 10, 27.5);
        BasicShape s2 = new Rectangle(Color.BLUE, 20, 20, 20, 50);
        BasicShape s3 = new Sphere(Color.BLUE, 30, 30, 30, 50);

        ArrayList<BasicShape> shapes = new ArrayList<>();
        shapes.add(s1);
        shapes.add(s2);
        shapes.add(s3);
        for (BasicShape shape : shapes) {
            System.out.println(shape);
        }
    }
}
```

Next week

- **Remedial Week**
- **Week after this is the Mock Exam 22/11/2023 at 1PM**
- **Come early please!**