JAVA PROGRAMMING II LAB

LAB # 5

Abstract Classes & Interfaces

Introduction

It makes sense to create a Rectangle object, a circle object or a square object, but what exactly is a Shape object? There is a lot of shapes but which one!!! So, the idea of the abstract classes appears. If we don't need to define an object of the class and its subclasses will creates objects then we define the class as an abstract class.

You can still use that abstract type as a reference type. In fact, that is a big part of why you have that abstract class in the first place (to use it as a polymorphic arguments or return type, or to make a polymorphic arrays). When you are designing your class inheritance structure, you have to decide which classes are abstract and which are concrete. Concrete classes are those that are specific enough to be instantiated. A concrete class just means that it is OK to make objects of that type.

Abstract Class

Abstract classes are like regular classes, but you cannot create instances of abstract classes using the new operator. It can contain abstract methods.

Abstract Method

An abstract method is defined in Abstract class without implementation. Its implementation is provided by the subclasses. A class that contains abstract methods must be defined abstract. The abstract methods are non-static and non-final.

Example of a shape class as an abstract class

```
abstract class Shape {

  public String color;
  public Shape() {
  }
  public void setColor(String c) {
    color = c;
  }
  public String getColor() {
    return color;
  }
  abstract public double area();
}
```

We can also implement the generic shapes class as an abstract class so that we can draw lines, circles, triangles etc. All shapes have some common fields and methods, but each can, of course, add more fields and methods. The abstract class guarantees that each shape will have the same set of basic properties. We declare this class abstract because there is no such thing as a generic shape. There can only be concrete shapes such as squares, circles, triangles etc.

```
class Point extends Shape {

 static int x, y;
 public Point() {
   x = 0;
   y = 0;
 }
 @Override
 public double area() {
   return 0;
 }
 public double perimeter() {
   return 0;
 }
 public static void print() {
   System.out.println("point: " + x + "," + y);
 } }


public class test {
      public static void main(String args[]) {
    Point p = new Point();
    p.print();
  }
```

The output is

```
point: 0,0
BUILD SUCCESSFUL (total time: 1 second)
```

Notice that, in order to create a Point object, its class cannot be abstract. This means that all of the abstract methods of the Shape class must be implemented by the Point class.

The subclass must define an implementation for every abstract method of the abstract superclass, or the subclass itself will also be abstract. Similarly other shape objects can be created using the generic Shape Abstract class. A big Disadvantage of using abstract classes is not able to use multiple inheritance. In the sense, when a class extends an abstract class, it can't extend any other class.

Java Interface

In Java, this multiple inheritance problem is solved with a powerful construct called interfaces. Interface can be used to define a generic template and then one or more abstract classes to define partial implementations of the interface. Interfaces just specify the method declaration (implicitly public and abstract) and can only contain fields (which are implicitly public static final). Interface definition begins with a keyword interface. An interface like that of an abstract class can't be instantiated.

Multiple Inheritance is allowed when extending interfaces i.e. One interface can extend none, one or more interfaces. Java does not support multiple inheritance, but it allows you to extend one class and implement many interfaces.

If a class that implements an interface does not define all the methods of the interface, then it must be declared abstract and the method definitions must be provided by the subclass that extends the abstract class.

```
interface Shape {

  public double area();
  public double volume();
}

   class Point implements Shape {

    static int x, y;
    public Point() {
      x = 0;
      y = 0;
    }
    public double area() {
      return 0;
    }
    public double volume() {
      return 0;
    }
    public static void print() {
      System.out.println("point: " + x + "," + y);
    }
    public static void main(String args[]) {
      Point p = new Point();
      p.print();
    }
}
```

Example

Our program shows 4 interfaces and 5 classes one being an abstract class.

Notes:

- The method toString() in class A is an overridden version of the method defined in the class named Object.
- The classes B, C and E satisfy the interface contract. But since the class D does not define all the methods of the implemented interface I2, the class D is declared abstract.
- Also, i1.methodI2() produces a compilation error as the method is not declared in I1 or any of its super interfaces if present. Hence a downcast of interface reference I1 solves the problem as shown in the program.
- The same problem applies to i1.methodA1(), which is again resolved by a downcast.
- When we invoke the toString() method which is a method of an Object, there does not seem to be any problem as every interface or class extends Object and any class can override the default toString() to suit your application needs.
- ((C)o1).methodI1() compiles successfully, but produces a ClassCastException at runtime. This is because B does not have any relationship with C except they are "siblings". You can't cast siblings into one another.
- When a given interface method is invoked on a given reference, the behavior that results will be appropriate to the class from which that particular object was instantiated. This is runtime polymorphism based on interfaces and overridden methods.
- Interface I4 inherits two interfaces and this allowed in Java.
- Class E isn't abstract because it implements the rest of unimplemented methods of its super classes.
- You can get the example by clicking here.