1.00/1.001
Recitation 06
Abstract Classes/Methods and Interfaces

March 19th & 20th 2012

Topics

- Abstract Classes (`extends`)
- Interfaces (`implements`)
- Polymorphism
- Problem Set 5

Abstract Classes: Content

- Some data members, like any class
- Some methods implemented (concrete)
- Some methods declared, but unimplemented (`abstract`)
  - We generally know what the method does
  - How the method performs may be different for different objects

Abstract Classes: Coding

- Abstract classes cannot be instantiated.
  - Instantiate (v.) – use the “new” keyword to create a new Object (or instance of a class)
  - Some methods remain unimplemented.
- Subclasses must implement all `abstract` methods, or must also be abstract classes.
- Why make a method abstract?
  - Provide some default behaviors in concrete methods
  - Programmer is FORCED to implement methods in a subclass before any object can be instantiated.

abstract Keyword

```
public abstract class MyClass {
        // data members
        private int myDataMember;

        public MyClass (int md){
          // concrete methods have 'bodies' or definitions
         myDataMember = md;
        }

        public int getData(){
         // concrete method
          return myDataMember;
        }

        public abstract int calc(int factor);
        // abstract methods omit the „body"
}
```

extends Keyword

```
public class AnotherClass extends MyClass{

      public AnotherClass (int md){
        // call constructor from “parent” or super class
        super(md);
      }

      // implement all abstract methods
      public int calc(int factor){
        return factor * factor;
      }
}
```

Abstract Classes: Exercise 1 p.1

1) Write an abstract class Shape
   - Data members: numSides
   - Constructor: initialize numSides
   - Concrete method: get method for numSides
   - Abstract methods: getArea(), getPerimeter()
2) Write a concrete subclass Rectangle
   - Data members: width, height
3) Write a concrete subclass RtTriangle
   - Data members: width, height
4) In another class, write a main method to define a Rectangle and a Triangle.

Solution: Shape

```
public abstract class Shape
{
    private int numSides;

    public Shape( int newSides)
        {numSides = newSides;}

    public int getNumSides()
        {return numSides;}

    public abstract double getArea();
    public abstract double getPerimeter();
}
```

Abstract Classes: Exercise 1 p.2

1) Write an abstract class Shape
   - Data members: numSides
   - Constructor: initialize numSides
   - Concrete method: get method for numSides
   - Abstract methods: getArea(), getPerimeter()
2) Write a concrete subclass Rectangle
   - Data members: width, height
3) Write a concrete subclass RtTriangle
   - Data members: width, height
4) In another class, write a main method to define a Rectangle and a Triangle.

Abstract Classes: Exercise p.3

1) Write an abstract class Shape
   - Data members: numSides
   - Constructor: initialize numSides
   - Concrete method: get method for numSides
   - Abstract methods: getArea(), getPerimeter()
2) Write a concrete subclass Rectangle
   - Data members: width, height
3) Write a concrete subclass RtTriangle
   - Data members: width, height
4) In another class, write a main method to define a Rectangle and a Triangle.

Abstract Classes: Exercise p.4

1) Write an abstract class Shape
   - Data members: numSides
   - Constructor: initialize numSides
   - Concrete method: get method for numSides
   - Abstract methods: getArea(), getPerimeter()
2) Write a concrete subclass Rectangle
   - Data members: width, height
3) Write a concrete subclass RtTriangle
   - Data members: width, height
4) In another class, write a main method to define a Rectangle and a Triangle.

Interfaces

- **“Its like a checklist”:** Class that **`implements`** an interface must implement/define all methods declared in the interface.
- A set of related method declarations.
- All method declarations omit the body.
- Constants may be defined.

- Why use interfaces?
  - Define a set of behaviors
  - Allow “multiple inheritance” by implementing multiple interfaces

Abstract Classes vs. Interfaces

- Abstract Classes have
  - Static and instance data members
  - Concrete and/or abstract methods
  - Single inheritance (via `extends`)
  - Constructor

- Interfaces have
  - Static final data members (constant)
  - All methods abstract
  - “Multiple Inheritance” (via `implements`)
  - No constructor

Remember Abstract Class Shape and Subclass Rectangle?

```
public abstract class Shape {
   private int numSides;
   public Shape(int numSides){
      this.numSides =
   numSides;
   }
   public double getNumSides()
   {
      return numSides; }
   public abstract double
   getArea();
   public abstract double
   getPerimeter();
}
```

```
public class Rectangle extends
       Shape {
   private double height, width;
   public Rectangle(double w,
            double h) {
       super(4);
       this.height = h;
       this.width = w;
   }
   public double getArea() {
       return height * width;
   }
   public double getPerimeter() {
       return 2 * (height + width);
   }
}
```

Interface: Exercise 2 p.1

1) Write an interface Resizable
   – Has a method `resize(double x)` that resizes a Shape's dimensions by factor x
2) Make Rectangle implement Resizable
3) Write a main method to:
   - Define a Rectangle (width = 2, height = 3)
   - Print the Rectangle"s area & perimeter
   - Resize the Rectangle by factor of 2
   - Re-print the Rectangle"s area & perimeter

Interface: Exercise 2 p.2

1) Write an interface Resizable
   - Has a method `resize(double x)` that resizes a Shape’s dimensions by factor x
2) Make Rectangle implement Resizable
3) Write a main method to:
   - Define a Rectangle (width = 2, height = 3)
   - Print the Rectangle‟s area & perimeter
   - Resize the Rectangle by factor of 2
   - Re-print the Rectangle‟s area & perimeter

Interface: Exercise 2 p.3

1) Write an interface Resizable
   - Has a method `resize(double x)` that resizes a Shape’s dimensions by factor x
2) Make Rectangle implement Resizable
3) Write a main method to:
   - Define a Rectangle (width = 2, height = 3)
   - Print the Rectangle‟s area & perimeter
   - Resize the Rectangle by factor of 2
   - Re-print the Rectangle‟s area & perimeter

`instanceof` Keyword

- The `instanceof` operator compares an object to a specified type.
- You can use it to test if an object is:
  - an instance of a class,
  - an instance of a subclass,
  - or an instance of a class that implements a particular interface.

Source: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html

instanceof Example

Here class **Lion and Cow extends Animal**

```
public class Animal {
   //body hidden
}
```

```
public class Cow extends Animal{
   //body hidden
}
```

```
public class Lion extends Animal{
   //body hidden
   public void roar(){//body hidden}
}
```

```
    public static void main(String[] args) {
        Animal[] zoo= new Animal[2];

        zoo[0] = new Cow();
        zoo[1] = new Lion();

        for( int i =0; i<zoo.length; i++){
            Animal a = zoo[i];
            if( a instanceof Lion){   //test using instanceof keyword
                System.out.println(”Animal " + i + “ is a Lion”);
                 Lion l = (Lion) a;    //Cast the Object to a Lion
                 l.roar();             //Call a method in the Lion class
            }
  }
```

Prints: `Animal 1 is a Lion`

Polymorphism: Exercise

- Write a main method
  - Create a `Rectangle` and a `RtTriangle`
  - Add them to an `ArrayList` of *`Shape`s*
  - Iterate through the Shapes in the ArrayList
    - If the Shape is Resizable, resize it by a factor of 0.5
    - Print out perimeter and area

Problem Set 5

- Write a program to model MBTA vehicles
- Three types of vehicles: Bus, Urban Rail, and Commuter Rail
- Three kinds of Right-of-Way: Dedicated, Shared, and Mixed (Hybrid)
- This homework tests your knowledge of inheritance. Your solution must inherit as much as possible from the superclasses and/or interfaces.
- Be sure to use at least one of EACH of the following in your solution: abstract class, interface, abstract method, final method.
- Hint: The trick is to determine if the set of Route Types and ROW Types should be Interfaces or Classes (Inheritance structure)
  - Which Types require "multiple inheritance"?