CS202 Java Object Oriented Programming

Advanced OOP Topics

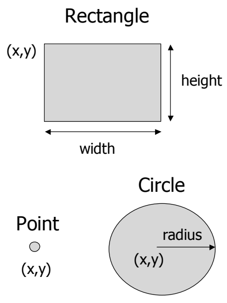
Chengyu Sun
California State University, Los Angeles

Overview

- Abstract Classes
- Multiple inheritance and Interfaces
- Nested classes

Shapes

- Attributes
  - Location
  - Length, width, Radius
- Operations
  - Move
  - Draw



Shape Class

```
public class Shape {

        protected int x, y; // initial location

    public Shape( int x, int y )
    {
        this.x = x;
        this.y = y;
    }

    public void move( int newX, int newY )
    {
        x = newX;
        y = newY;
    }

    public void draw() { ??? }
}
```

Abstract Shape Class

- An abstract class
  - Some operations are known and some are not
  - Unknown operations can be declared as abstract methods
  - Cannot be instantiated

```
public abstract class Shape {

        int x, y; // location

    public Shape( int x, int y )
    {
        this.x = x;
        this.y = y;
    }

    void move( int newX, int newY )
    {
        x = newX;
        y = newY;
    }

    public abstract void draw();
}
```

Subclasses of Shape

- Point, Rectangle, and Circle
- A concrete class
  - A subclass of an abstract superclass
  - Must implement (override) the abstract methods
  - Can be instantiated
- Why do we need a superclass when there’s so little code reuse??

Sort Integers

```
public void sort( int a[] )
{
   int left = 0;
   while( left < a.length-1 )
   {
      int index = left;
      for( int i=left ; i < a.length ; ++i )
         if( a[i] < a[index] ) index = i;

      // swap a[index] and a[left]
      int tmp = a[index];
      a[index] = a[left];
      a[left] = tmp;

      ++left;
   }
}
```

Sort Objects

- Any objects that has a lessThan() method

```
public abstract class Comparable {

   public Comparable() {}

   // return true if this object is less than o
   public abstract boolean lessThan( ?? o );
}
```

A More General Sort

```
public void sort( Comparable a[] )
{
   int left = 0;
   while( left < a.length-1 )
   {
      int index = left;
      for( int i=left ; i < a.length ; ++i )
         if( a[i].lessThan(a[index]) ) index = i;

      // swap a[index] and a[left]
      int tmp = a[index];
      a[index] = a[left];
      a[left] = tmp;

      ++left;
   }
}
```

The Need for Multiple Inheritance

- What if we want to sort an array of `Point`?
  - Inherit both Shape *and* Comparable?

The Problem of Multiple Inheritance

```
public class A {          public class B {          public class C extends A, B
                                                    {
… …                       … …                          … …
     public int x;             public int x;        }

  public void foobar()      public void foobar()
  {                         {
     …                         …
  }                         }

}                         }
```

- Which x or `foobar()` does C inherit?

Interface

- Java's answer to multiple inheritance
- A interface only contains
  - Method declarations
    - No method implementations
    - All methods are implicitly `public` and `abstract`
  - Constants
    - All constants are implicitly `public`, `static`, and `final`

Interface Examples

```
public interface ActionListener
{
    public void actionPerformed(ActionEvent ae);
}

public interface AdjustmentListener
{
    public void adjustmentValueChanged(AdjustmentEvent e);
}

public interface MouseListener
{
    public void mousePressed();
    public void mouseClicked();
    public void mouseReleased();
    public void mouseEntered();
    public void mouseExited();
}
```

`Comparable` Interface

```
public interface Comparable {

    boolean lessThan( Object c );

}
```

Interface Usage

```
public class Point extends Shape implements Comparable {

    public Point( int x, int y ) { super(x,y); }

    public void draw() { … }

    public boolean lessThan( Object o )
    {
        Point p = (Point) o; // cast to a Point for comparable
        ??
    }

 } // end of class Point
```

Exercise: Interface Constants

```
public interface InterA {

    final int x = 10;

    void print();
}

public interface InterB {

    final int x = 20;

    void print();
}
```

```
public class C implements InterA, InterB {

    public void print()
    {
        System.out.println(x);
    }

    public static void main( String args[] )
    {
        C c = new C();
        c.print();
    }
}
```

Exercise: Interface Constants

```
public interface InterA {

    final int x = 10;

    void print();
}

public interface InterB {

    final int x = 20;

    void print();
}
```

```
public class C implements InterA, InterB {

    void print()
    {
        System.out.println(x);
    }

    public static void main( String args[] )
    {
        C c = new C();
        c.print();
    }
}
```

- Try run the code above, observe the error, and correct it

Abstract Class vs. Interface

- Abstract class
  - An incomplete class
  - Class variables
  - Constructors
  - Methods and abstract methods
  - `extends`
  - Single inheritance
  - Cannot be instantiated
- Interface
  - Not a class at all
  - Only constants
  - No constructors
  - Only abstract methods (method declarations)
  - `implements`
  - Multiple implementation
  - Cannot be instantiated

Nested Classes

- A class inside another class

```
pubic class A {
    …
    // a nested class
    class B { … }
}
```

Simple Nested Class Example

- ArrayWrapper and Iterator
  - hasMoreElements()
  - nextElement()

Properties of Nested Class

- Can access all members of the outer class, including `private` members
- Type
  - Inside outer class: InnerClassName
  - Outside outer class: OuterClassName.InnerClassName
- Can be declared as `public`, `protected`, or `private`
- Can be `static` or non-static (inner class)