Polymorphism: Interfaces and Iteration

Fundamentals of Computer Science
• **A shape object hierarchy**
  - Classes that *extend*
  - Versus classes that *implements*

• **Java interfaces**
  - How Java handles multiple-inheritance
  - *A promise to implement a set of methods*
    - Implementation left to class, all abstract methods

• **Iteration**
  - Moving through elements of a collection
  - In Java, class must implement iterable interface
Shape Object Hierarchy

- **Represent shapes that can:**
  - Draw themselves
  - Test for intersection with \((x, y)\) coordinate
  - Change color
  - Support:
    - Circles
    - Rectangles
    - Circles with borders
    - Rectangles with borders
1) Can we create a Shape object?
2) Which classes are abstract?
3) Which classes are concrete?
4) Which are subclasses of Shape?
5) Which are subclasses of Circle?
6) Which methods are overridden?
7) Which methods are overloaded?
Shapes with Borders

- **CircleBorder and RectangleBorder**
  - Share two identical instance variables
  - Share four identical methods
  - We’d like to consolidate to avoid repeated code
Option 1: Move into Shape Base Class

```java
public abstract class Shape {
    private double x, y;
    private Color color;
    private double thickness;
    private Color borderColor;

    public double getX();
    public double getY();
    public Color getColor();
    public void setColor(Color color);
    public double distance(double x, double y);
    public abstract void draw();
    public abstract boolean intersects(double x, double y);

    public Color getBorderColor();
    public void setBorderColor(Color color);
    public double getBorderThickness();
    public void setBorderThickness(double thickness);
}
```

This works and does share implementation, but:

Now Circle and Rectangle also have instance variables and methods related to having a border.

They have an API that lies.

```java
public class Circle extends Shape {
    private double radius;
    public double getRadius();
    public void draw();
    public boolean intersects(double x, double y);
}
```

```java
public class Rectangle extends Shape {
    private double width, height;
    public double getWidth();
    public void draw();
    public boolean intersects(double x, double y);
}
```

```java
public class CircleBorder extends Circle {
    public void draw();
}
```

```java
public class RectangleBorder extends Rectangle {
    public void draw();
}
```
Option 2: Multiple Inheritance

This does **NOT** work.

Java does not support multiple inheritance.

You can only extend a single class.
Java Interfaces

• **Java's alternative to multiple inheritance**
  - Classes promise to implement same API
    - An interface is just a list of abstract methods
    - If two classes implement the same interface, they can live in the same array for polymorphic goodness
  - Example uses:
    - Allow any object type to be sorted
    - For-each (enhanced) loops
    - GUI event listeners
      - e.g. When a button is pushed
    - Classes that can run in their own thread
public class CircleBorder extends Circle implements Bordered

// Interface for a shape that has a border that has a different color and has a variable pen thickness.

public interface Bordered
{
    Color getBorderColor();
    void setBorderColor(Color color);
    double getBorderThickness();
    void setBorderThickness(double thickness);
}

A class adds implements Bordered to the class declaration.
The class must then implement the four methods in interface Bordered.

public class CircleBorder extends Circle implements Bordered

public class RectangleBorder extends Rectangle implements Bordered
GrowShape

- **Show a bunch of Shape objects**
  - If mouse is over a shape:
    - Temporarily change object's color
    - If object has a border, permanently grow the border
```java
public static void main(String [] args)
{
    Shape [] shapes = new Shape[4];

    shapes[0] = new RectangleBorder(0.5, 0.5, 0.1, 0.2);
    shapes[1] = new CircleBorder(0.2, 0.2, 0.15);
    shapes[2] = new Circle(0.9, 0.9, 0.1);
    shapes[3] = new Rectangle(0.9, 0.2, 0.2, 0.2);

    while (true)
    {
        StdDraw.clear();
        double x = StdDraw.mouseX();
        double y = StdDraw.mouseY();
        for (Shape shape : shapes)
        {
            if (shape.intersects(x, y))
            {
                shape.setColor(new Color(0.3f, 0.1f, 0.5f));
                if (shape instanceof Bordered)
                {
                    Bordered bordered = (Bordered) shape;
                    double currentThickness = bordered.getBorderThickness();
                    bordered.setBorderThickness(currentThickness + 0.001);
                }
            }
            else
            {
                shape.setColor(new Color(0.0f, 0.0f, 1.0f));
                shape.draw();
            }
        }
        StdDraw.show(100);
    }
}
```

Polymorphic array holding objects in the Shape hierarchy. Some have borders, some don't.

Only increase the border on objects that implements the Bordered interface.

You must check using `instanceof` before casting object.
Loops over Things in a Collection

- **Enhanced for-loop**
  - Move through all items in many data types
    - e.g. arrays, ArrayList, HashSet, Stack, LinkedList
  - Caller doesn't know how items stored
  - Requires type implement the `iterable` interface

- **Many Collections in Java are already iterable**
  - [http://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html](http://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html)
Examples of Iteration with Built-In Classes

```java
String [] namesA = new String[2];
namesA[0] = "Bob";
namesA[1] = "Abe";
for (String s: namesA)
    System.out.println("array: " + s);

ArrayList<String> namesB = new ArrayList<String>();
namesB.add("Bob");
namesB.add("Abe");
for (String s: namesB)
    System.out.println("ArrayList: " + s);

HashSet<String> namesC = new HashSet<String>();
namesC.add("Bob");
namesC.add("Abe");
for (String s: namesC)
    System.out.println("HashSet: " + s);

Stack<String> namesD = new Stack<String>();
namesD.push("Bob");
namesD.push("Abe");
for (String s: namesD)
    System.out.println("Stack: " + s);

LinkedList<String> namesE = new LinkedList<String>();
namesE.add("Bob");
namesE.add("Abe");
for (String s: namesE)
    System.out.println("LinkedList: " + s);
```

The order of iteration through objects depend on the collection and its implementation.
To make a collection iterable:
- Must implement an `iterator()` method that returns an `Iterator` object
- The `Iterator` class must include two methods:
  - `hasNext()` - Returns a `boolean` indicating if there are more items
  - `next()` - Returns an item from the collection
- [http://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html](http://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html)

Create some iterable collection.

Normal enhanced for-loop version printing out all the elements.

Using iterator object to explicitly loop over all elements in collection.
Iterators Implement an Interface

```java
public interface Iterator<Item>
{
    boolean hasNext();
    Item next();
    void remove();
}
```

"Removes from the underlying collection the last element returned by the iterator (optional operation)."

- Java API

"Interleaving iteration with operations that modify the data structure is best avoided."

- Robert Sedgewick, Kevin Wayne

- **Iterable collections return an **Iterator** object**
  - Object tracks where we are within the collection
  - For a collection backed by an array → integer index
  - For a collection backed by a linked list → a pointer
public class RandomShapes {
    private Shape [] shapes;

    public RandomShapes(int num) {
        if (num <= 0)
            return;
        shapes = new Shape[num];
        for (int i = 0; i < num; i++) {
            double x = Math.random();
            double y = Math.random();
            if (Math.random() < 0.5)
                shapes[i] = new Circle(x, y, Math.random() * 0.1);
            else
                shapes[i] = new Rectangle(x, y, Math.random() * 0.1, Math.random() * 0.1);
        }
    }

    public static void main(String [] args) {
        RandomShapes shapes = new RandomShapes(20);
        for (Shape s : shapes)
            s.draw();
    }
}

Collection of Random Shapes
Can only iterate over an array or an object that implements java.lang.Iterable
import java.util.Iterator;

public class RandomShapes implements Iterable<Shape> {
    private Shape [] shapes;

    public Iterator<Shape> iterator() {
        return new ArrayIterator();
    }

    private class ArrayIterator implements Iterator<Shape> {
        private int i = 0;
        public boolean hasNext() { return i < shapes.length; }
        public Shape next() { return shapes[i++]; }
        public void remove() {
        }
    }
    ...
}
Summary

- **Shape object hierarchy**
  - Circle and Rectangle branches
  - CircleBorder and RectangleBorder
    - Share functionality with each other, but not with parent class

- **Java interfaces:** Promise to implement an API
  - Objects unrelated by inheritance can live in same array
  - A class can implement multiple interfaces
  - Used by Java for sorting objects and much more

- **Iteration:** Move through a collection’s objects
  - Without knowledge of underlying data structure
  - Implements the iterable interface