CSE 143
Lecture 20

Abstract classes

Circle

```
public class Circle {
   private double radius;

   public Circle(double radius) {
      this.radius = radius;
   }

   public double area() {
      return Math.PI * radius * radius;
   }

   public String toString() {
      return "circle of area " + area();
   }
}
```

Rectangle

```
public class Rectangle {
    private double length;
    private double width;

   public Rectangle(double length, double  width) {
        this.length = length;
        this.width = width;
    }

    public double area() {
        return length * width;
    }

    public String toString() {
        return "rectangle of area " + area();
    }
}
```

3

Square

```
public class Square {
    private double length;

    public Square(double length) {
        this.length = length;
    }

    public double area() {
        return length * length;
    }

    public String toString() {
        return "square of area " + area();
    }
}
```

4

Shape client code

```
import java.util.*;

public class ShapeTest {
   public static void main(String[] args) {
      Object[] data =
       {new Square(12), new Rectangle(15, 3.2),
        new Circle(8.4), new Circle(1.5), new Square(8.7),
        new Rectangle(7.2, 3.2), new Square(2.4),
        new Circle(3.7), new Circle(7.9)};

      for (Object s : data) {
         System.out.println(s);
      }
      System.out.println();
      Arrays.sort(data);
      for (Object s : data) {
         System.out.println(s);
      }
   }
}
```

Recall that the Object class is at the top of the inheritance hierachy

5

Comparable?

- Client code crashes when trying to sort the elements

```
Exception in thread "main" java.lang.ClassCastException:
Square cannot be cast to java.lang.Comparable
            at java.util.Arrays.mergeSort(Arrays.java:1144)
            at java.util.Arrays.mergeSort(Arrays.java:1155)
            at java.util.Arrays.sort(Arrays.java:1079)
            at ShapeTest.main(ShapeTest.java:13)
```

- Sort method tries to cast object to Comparable so it can call `compareTo` method.
- BTW, notice that Java uses merge sort to sort!

6

Implementing Comparable

```
public class Square implements Comparable<Square> {
    ...
```

- Writing the `compareTo` method…

```
public int compareTo(Square other) {
    return area() - other.area();
}
```

- The previous method has the wrong return type:

```
public int compareTo(Square other) {
    return (int)(area() - other.area());
}
```

7

Writing compareTO

- This previous method also fails! If the difference is between -1 and 1. Casting to `int` rounds it to 0.
- Correct answer:

```
public int compareTo(Square other) {
    double difference = area() - other.area();
    if (difference < 0) {
      return -1;
    } else if (difference == 0) {
       return 0;
    } else { // difference > 0
       return 1;
    }
 }
```

8

Client code still crashes…

```
Exception in thread "main" java.lang.ClassCastException:
Rectangle cannot be cast to Square
            at Square.compareTo(Square.java:1)
            at java.util.Arrays.mergeSort(Arrays.java:1144)
            at java.util.Arrays.mergeSort(Arrays.java:1155)
            at java.util.Arrays.sort(Arrays.java:1079)
            at ShapeTest.main(ShapeTest.java:13)
```

- Differs from previous error message: `Square cannot be cast to java.lang.Comparable`
- Currently Squares and Rectangles are not related to each other.

9

Shape interface

```
public interface Shape {

}

public class Square implements Comparable<Shape> {
 ...
   public int compareTo(Shape other) {
      ...
   }


public class Rectangle implements Comparable<Shape> {
 ...

public class Circle implements Comparable<Shape> {
 ...
```

10

Area undefined?

```
Square.java:5: cannot find symbol
     symbol  : method area()
     location: interface Shape
```

- Notice the "location".

```
public interface Shape {
    public double area();
}
```

11

More ClassCastExceptions!

```
Exception in thread "main" java.lang.ClassCastException:
Rectangle cannot be cast to Shape
            at Square.compareTo(Square.java:1)
            at java.util.Arrays.mergeSort(Arrays.java:1144)
            at java.util.Arrays.mergeSort(Arrays.java:1155)
            at java.util.Arrays.sort(Arrays.java:1079)
            at ShapeTest.main(ShapeTest.java:13)
```

- We never said that Rectangles were Shapes…

```
public class Rectangle implements Shape, Comparable<Shape> {
```

- But all Shapes are to be Comparable anyway… Notice that interfaces extend other interface (not implements!)

```
public interface Shape extends Comparable<Shape> {
public class Rectangle implements Shape {
```

12

Modified client code

```
import java.util.*;

public class ShapeTest {
   public static void main(String[] args) {
      Shape[] data =
       {new Square(12), new Rectangle(15, 3.2),
        new Circle(8.4), new Circle(1.5), new Square(8.7),
        new Rectangle(7.2, 3.2), new Square(2.4),
        new Circle(3.7), new Circle(7.9)};

      for (Shape s : data) {
         System.out.println(s);
      }
      System.out.println();
      Arrays.sort(data);
      for (Shape s : data) {
         System.out.println(s);
      }
   }
}
```


Removing redundancy

- Each shape class has the exact same `compareTo` method!

```
public interface Shape extends Comparable<Shape> {
    public double area();

    public int compareTo(Shape other) {
       double difference = area() - other.area();
       if (difference < 0) {
         return -1;
       } else if (difference == 0) {
          return 0;
       } else { // difference > 0
          return 1;
       }
    }
}
```

Interfaces and methods

- Methods in interfaces cannot have bodies.

```
Shape.java:4: interface methods cannot have body
            public int compareTo(Shape other) {
                                                ^
    1 error
```

- Change to class and implement Comparable

```
public class Shape implements Comparable<Shape> {
   public double area();

   public int compareTo(Shape other) {
      double difference = area() - other.area();
      if (difference < 0) {
       return -1;
      } else if (difference == 0) {
         return 0;
      } else { // difference > 0
         return 1;
      }
   }
}
```

15

Missing method body

```
  Shape.java:2: missing method body, or declare abstract
            public double area();
                          ^
    1 error
```

- What about this?

```
   public double area() {
      return 0; // dummy value
   }
```

- Hacks are never good ideas… Declare class as abstract.

```
public abstract class Shape implements Comparable<Shape> {
   public abstract double area();

   public int compareTo(Shape other) {
      ...
   }
}

public class Rectangle extends Shape { // Shape is no longer an interface
```

16

Continuum of classes

- Concrete class
  - All methods defined
- Abstract class
  - Some methods defined
- Interface
  - No methods defined

17

Abstract and interfaces

- Normal classes that claim to implement an interface must implement all methods of that interface:

```
public class Empty implements Shape {}  // error
```

- Abstract classes can claim to implement an interface without writing its methods; subclasses must implement the methods.

```
public abstract class Empty implements Shape {} // ok
public class Child extends Empty {}       // error
```

18

Object creation

- Since abstract classes have some undefined methods, they cannot be instantiated. (Recall that interfaces cannot be instantiated too!)

```
Shape s = new Shape();  // illegal

Shape s = new Rectangle(20, 30);  // ok!
```

19

Abstract class vs. interface

- Why do both interfaces and abstract classes exist in Java?
  - An abstract class can do everything an interface can do and more.
  - So why would someone ever use an interface?
- Answer: Java has single inheritance.
  - can extend only one superclass
  - can implement many interfaces
  - Having interfaces allows a class to be part of a hierarchy (polymorphism) without using up its inheritance relationship.

20

Last redundancy

- toString() method is a bit redundant
- Put name and toString() in abstract superclass:

```
public abstract class Shape implements Comparable<Shape> {
  private String name;

  public Shape(String name) {
     this.name = name;
  }

  public abstract double area();

  public int compareTo(Shape other) { ... }

  public String toString() {
     return name + " of area " + area();
  }

}
```

21

New constructors

```
public Circle(double radius) {
    super("cicle");
    this.radius = radius;
 }

public Rectangle(double length, double  width) {
    super("rectangle");
    this.length = length;
    this.width = width;
 }

public Square(double length) {
    super("square");
    this.length = length;
 }
```

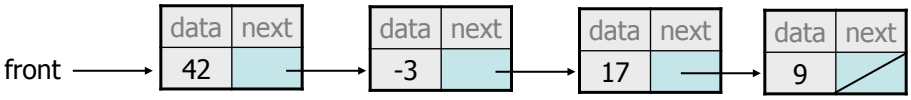
22

Linked vs. array lists

- We have implemented two collection classes:
  - `ArrayIntList`

index	0	1	2	3
value	42	-3	17	9

  - `LinkedIntList`



  - They have similar behavior, implemented in different ways. We should be able to treat them the same way in client code.

An `IntList` interface

```
// Represents a list of integers.
public interface IntList {
    public void add(int value);
    public void add(int index, int value);
    public int get(int index);
    public int indexOf(int value);
    public boolean isEmpty();
    public boolean contains(int value);
    public void remove(int index);
    public void set(int index, int value);
    public int size();
}
```

```
public class ArrayIntList implements IntList { ...
public class LinkedIntList implements IntList { ...
```

An abstract list class

```
// Superclass with common code for a list of integers
public abstract class AbstractIntList implements IntList {
    public void add(int value) {
        add(size(), value);
    }

    public boolean contains(int value) {
        return indexOf(value) >= 0;
    }

    public boolean isEmpty() {
        return size() == 0;
    }
}


public class ArrayIntList extends AbstractIntList { …

public class LinkedIntList extends AbstractIntList { …
```

25

Putting it all together

```
// Superclass with common code for a list of integers
public abstract class AbstractList<E> implements List<E> {
    public void add(E value) {
        add(size(), value);
    }

    public boolean contains(E value) {
        return indexOf(value) >= 0;
    }

    public boolean isEmpty() {
        return size() == 0;
    }
}


public class ArrayList<E> extends AbstractList<E> { …

public class LinkedList<E> extends AbstractList<E> { …
```

See: http://download.oracle.com/javase/6/docs/api/java/util/AbstractList.html

26