Advanced Programming Techniques in Java

In

- In
- A

Review: Abstract Classes

- An abstract class is a placeholder in a class hierarch

  An abstract class cannot be instantiated

- Why?
- The use of abstract classes is a design decision; it helps
  are too general to instantiate

Review: Abstract Classes

- To declare a class as abstract we use the modifier `ab` header
- **Syntax**

```
public abstract class <name>
{ // contents }
```

- **Example**

```
public abstract class
Shape{ // contents }
```

- If the client code tries to create a `Shape` object, we

`Cannot instantiate the type Shape`

Review: Abstract Method

- An abstract method is a method that has just the sign

```
public abstract <type> <name>(<type> <n
```

- A class declared as `abstract` **does not** need to cor

Review: Abstract Classes & Constru

- Can an abstract class have a constructor?
- An abstract class can have a constructor. You can either class or if you don't, the compiler will add a default const

- Why can an abstract class have a constructor?
- When a class extends an abstract class, the constructor super class either implicitly or explicitly

Review: Multiple Inheritance

- Java supports single inheritance, meaning that a de
  class
- Multiple inheritance allows a class to be derived fro
  members of all parents
- **Java does not support multiple inheritance** beca

conflicts
  - Which class should `super` refer when child cl
- Alternative: **Interface**
  - Looks like a class
  - It describes what a class does

- You can have a class that **extends** one class an

Review: Interface Definition

FORM:

```
public interface interfaceName { abstract
    method headings constant declarations
}
```

EXAMPLE:

```
public interface Payable { public abstract
    double calcSalary(); public abstract
    boolean salaried(); public static final
    double DEDUCTIONS = 25.5;
}
```

constant declaration

- As such, they may be omitted

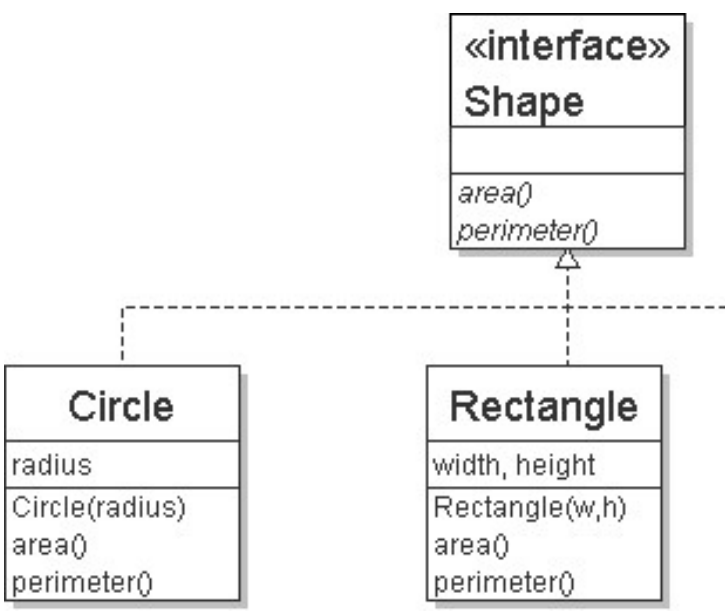
Shape Interface

- An interface for shapes:

```
// A general interface for shape cla
public interface Shape {
    public double area();
    public double
    perimeter();
}
```

- This interface describes the features common
area and perimeter)

Diagrams of Interfaces



- We draw arrows upward from the classes to

- There is a supertype-subtype relationship here: Shapes are Circles

Complete Circle class

```
// Represents circle shape
public class Circle implements Shap
    private double radius;

    // Constructs a new circle with
    public Circle(double radius) {
        this.radius = radius;
    }

    // Returns the area of this ci
    public double area() { return M
    * radius * radius;
    }

    // Returns the perimeter of thi
    public double perimeter() { ret
    2.0 * Math.PI * radius; }
}
```

Complete Rectangle class

```
// Represents rectangle shapes.
public class Rectangle implements Sha
    private double width; private dou
    height;

    // Constructs a new rectangle wit
    public Rectangle(double width, do
        this.width = width; this.heig
    }

    // Returns the area of this recta
    public double area() { return wid
    height;
    }

    // Returns the perimeter of this
    public double perimeter() { retur
    (width + height); }
}
```

Complete Triangle class

```
// Represents triangle shapes. public
class Triangle implements Shape {
private double a; private double b;
private double c;

    // Constructs a new Triangle given side
    public Triangle(double a, double b, dou
        { this.a = a; this.b = b; this.c =
    }

    // Returns this triangle's area using H
    public double area() { double s
        = (a + b + c) / 2.0;
        return Math.sqrt(s * (s - a) * (s -
    }

    // Returns the perimeter of this
    triangle. public double perimeter() {
    return a + b + c;
    }
}
```

Inheriting from Interf

- A class can *extend* 0 or 1 superclass
- An interfacecannot extend
- A class can *implement* c more interfaces
- An interfacecan extend c more interfaces

Summary of Features of Abstract Classes,and

Property	Act
Instances (objects) of this can be created.	
This can define instance variables and methods.	
This can define constants.	
The number of these a class can extend.	
The number of these a class can implement.	
This can extend another class.	
This can declare abstract methods.	
Variables of this type can be declared.	

ADTs

- Abstract Data Type (ADT)
- An encapsulation of data and methods
- Allows for reusable code
- The user need not know about the implementation of the ADT

- A user interacts with the
  only public methods

Problem

- Write a program that reads a file and displays:
- First display all words
- Then display them with all plurals capitalized
- Then display them in reverse order
- Then display them with all plural words removed

Naive solution

```
String[] allWords = new String[1000];
int wordCount = 0;

Scanner input = new Scanner(new
File("data.txt")); while (input.hasNext(
String word = input.next(); allWords[wor
word; wordCount++;
}
```

- You don't know how many words the file will ha
- Hard to create an array of the appropriate size
- Luckily, there are other ways to store data bes

Array Limitations

- You need to know in advance the maximum nu
  changes later?

- What if you want to remove something?
- You end up with empty elements in the middle of the
- You would have to shift the rest of the elements ove

- We need more flexibility!!
- “Add something here to the list”

- “Remove this value from the list”
- List should grow and shrink and move elements

ArrayLis

The `ArrayList` class

- An `ArrayList` object uses an array to store its
- Think of it as an auto-resizing array that can hol convenient methods
- It maintains most of the benefits of arrays, such

- It frees us from some tedious operations on arrays,
- To use `ArrayList` remember to import `java.ut`
- We can declare arrays of different types e.g., `int[` class has similar flexibility

The `ArrayList` class

- `ArrayList<E>` is a **generic class**
- The `<E>` is a placeholder in which you write the type

ArrayList

Java Generics

- Used to make an object usable for any types, while
  Java allows
- Normally we must be specific about the type we're
  allows us to make this variable
- Useful for making data structures, which we want to
  to insert into them
- We want to create a "box" that allows you to put so
  in there
- We could make a box for `Points`, and then copy th

Java Generics

```
public class PointBox{
   private Point p; publi
   void put(Point p){
   this.p = p;
   }
   public Point get( ){
      return this.p;
   }
}
```

```
public static
      PointBo
      pb1.put
      System.
}
```

- We can make this code “generic”

Java Generics

```
public class PointBox{
    private Point p; public
    void put(Point p){ this.p
    = p;
    }
    public Point get( ){
        return this.p;
    }
}
```

```
publ
    T
    p
    =
    }
    p

    }
}
```

- Now we can put an object of any type “T” into the b

How to use this “Generic” Type

- In the `main` method, you can initialize a `B`

```
Box<TYPE> name = new Box<TYPE>( );
```

- e.g: `Box<String> stringBox = new Box`
- or: `Box<Point> pointBox = new Box<Po`

used for any type*!

Example Code

```
public class Main{
  public static void main(String[ ] a
    Point p2 = new Point(0,5);
    System.out.println("Making a box
    Box<Point> b1 = new Box<Point>( )
    b1.put(p2);
    System.out.println(b1.get().getY

  }
}
```

Makes a

Java doesn't complain that we do `.getY()` on th
the box, since we told it that the object was going

In summary …

- **Generic class** is a type in Java that is written to
- Generic (or "parameterized") classes were added to J
  type safety of Java's collections
- A parameterized type has one or more other types' na

Why Use Generic Collect

- Better type-checking: catch more e

```
// without Generics
List list = new ArrayList();
list.add("hello");

// With Generics
List<Integer> list = new Arr
list.add("hello"); // will n
```

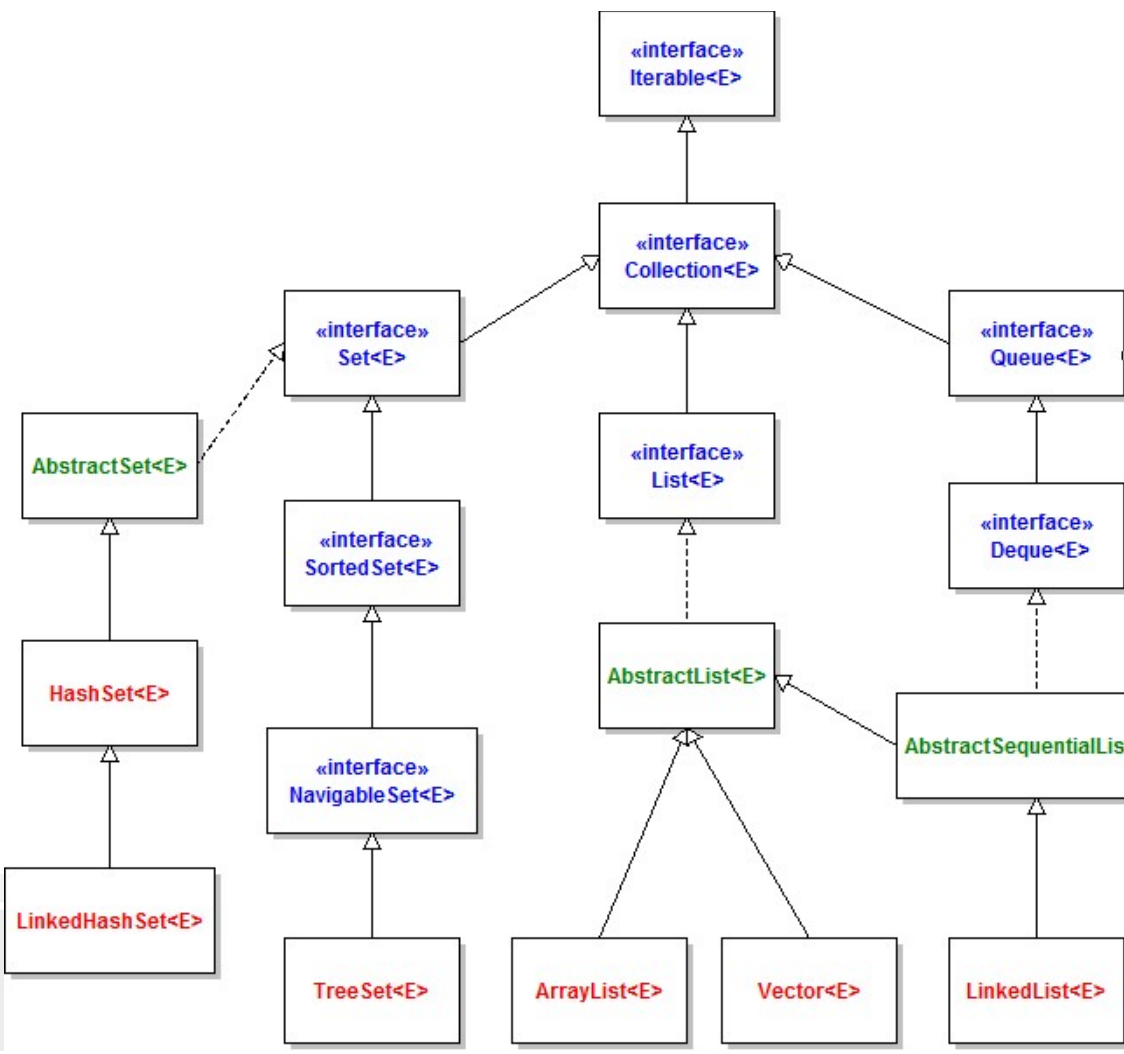
  them earlier
- Documents intent
- Avoids the need to downcast from

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

When re-written to use generics, the code doe

```
List<String> list = new ArrayList<Stri
list.add("hello");
String s = list.get(0); // no cast
```

Overview of Java Collections Fram



```
public class ArrayList<E> extends AbstractL
```

Java collections framework