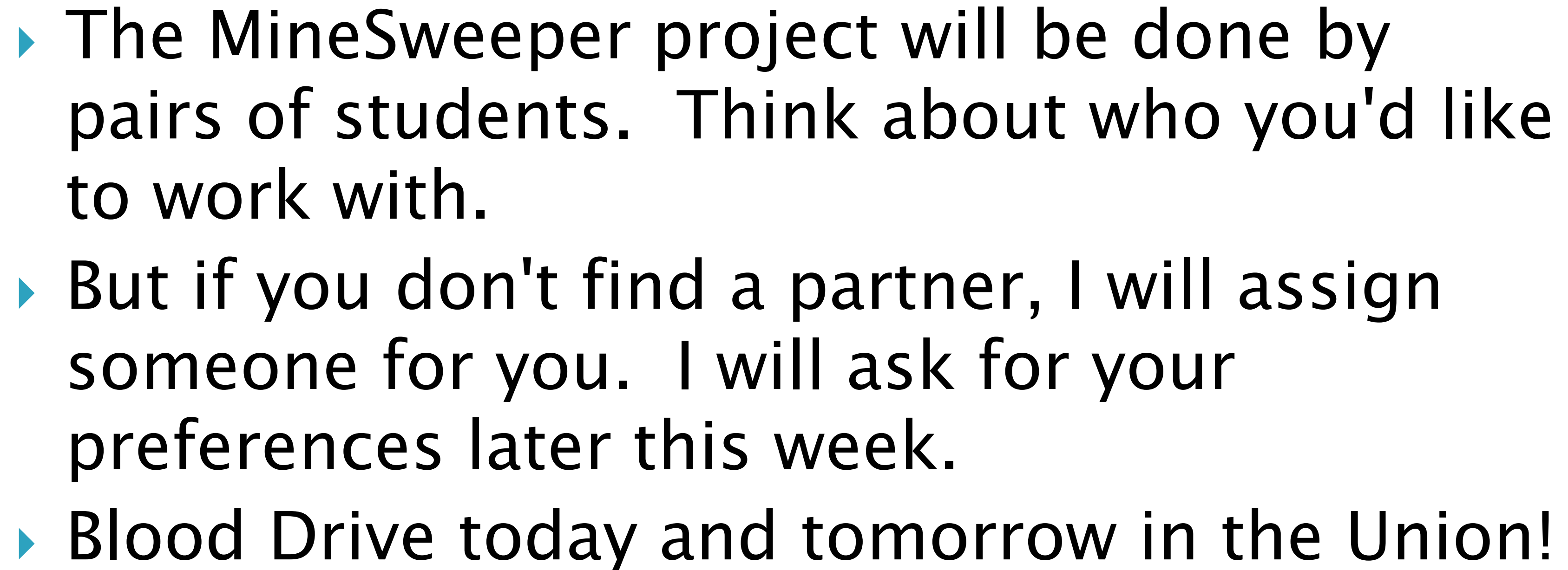
CSSE 220 Day 5

Subversion
Inheritance

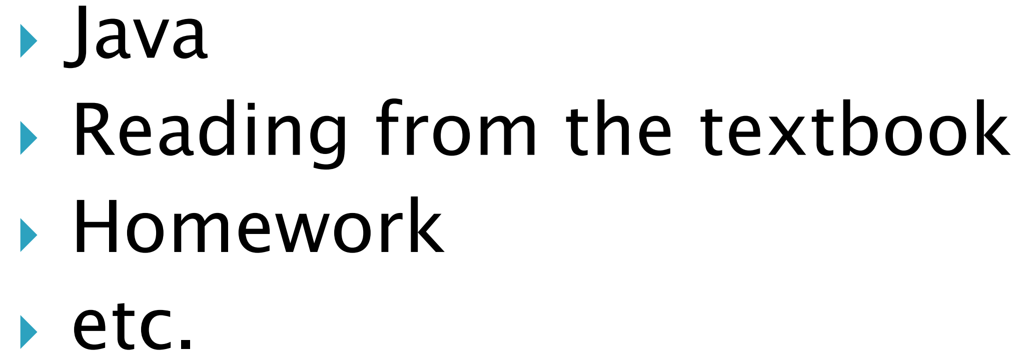
CSSE 220 Day 5



Poll results

Result Summary		
0	0%	Much too slow
3	8%	A little too slow
19	48%	About right
15	38%	A bit too fast
3	8%	I'm mostly lost during class discussion

Your questions about …



- Java
- Reading from the textbook
- Homework
- etc.

Recap: "Resize" an array

- An array is inherently fixed-length.
- But we can get the effect of a "growable array":
  - Have two variables, arr, and size.
  - initialize arr to be an array of 5 elements
    - I choose 5 because that is what Mark Weiss does.
  - When we want to add a new element at the end:
    - if size == arr.length
      - call resize to give us an array twice as big.
    - Put the new element in arr[size] and increment size.
    - Code:

```
if (size == arr.length)
   arr = resize(arr, size, size*2);
arr[size++] = newValue;
```

Write resize( )

Why *2 instead of +1?
You'll answer that question mathematically on the first day of 230 (if not sooner)

resize Solution

```
int[] resize(int[] a, int oldsize, int newsize){
  int[] result = new int[newSize];
  int numToCopy = Math.min(oldsize, newsize);

  for (int i=0; i < numToCopy; i++) {
     result[i] = a[i];

  return result;
}
```

ArrayList: a class that implements a resizeable array-like structure

- Full name: **java.util.ArrayList**
- Methods include
  - add(element)
  - add(index, element)
  - get(index)
  - size( )
  - clear( )
  - remove(object)
  - remove(index)
  - set(index, element)
  - toArray( )
  - trimToSize( )

Version Control Systems (review)

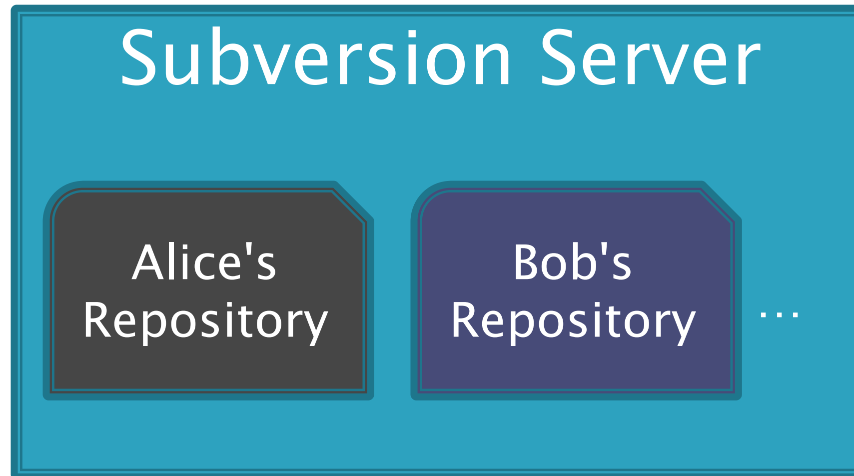
- Store "snapshots" of all the changes to a project over time
- Benefits:
  - Allow multiple users to share work on a project
  - Act as a "global undo"
  - Record who made what changes to a project
  - Maintain a log of the changes made
  - Can simplify debugging
  - Allow engineers to maintain multiple different versions of a project simultaneously

Our Version Control System

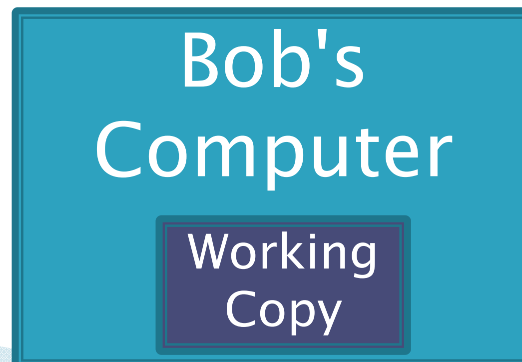
- Subversion, sometimes called SVN
- A free, open-source application
- Lots of tool support available
  - Works on all major computing platforms
  - TortoiseSVN for version control in Windows Explorer
  - Subclipse for version control inside Eclipse

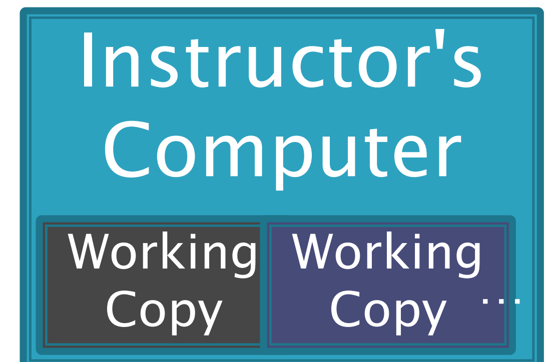
Version Control Terms

Repository: the copy of your data on the server, includes *all* past versions

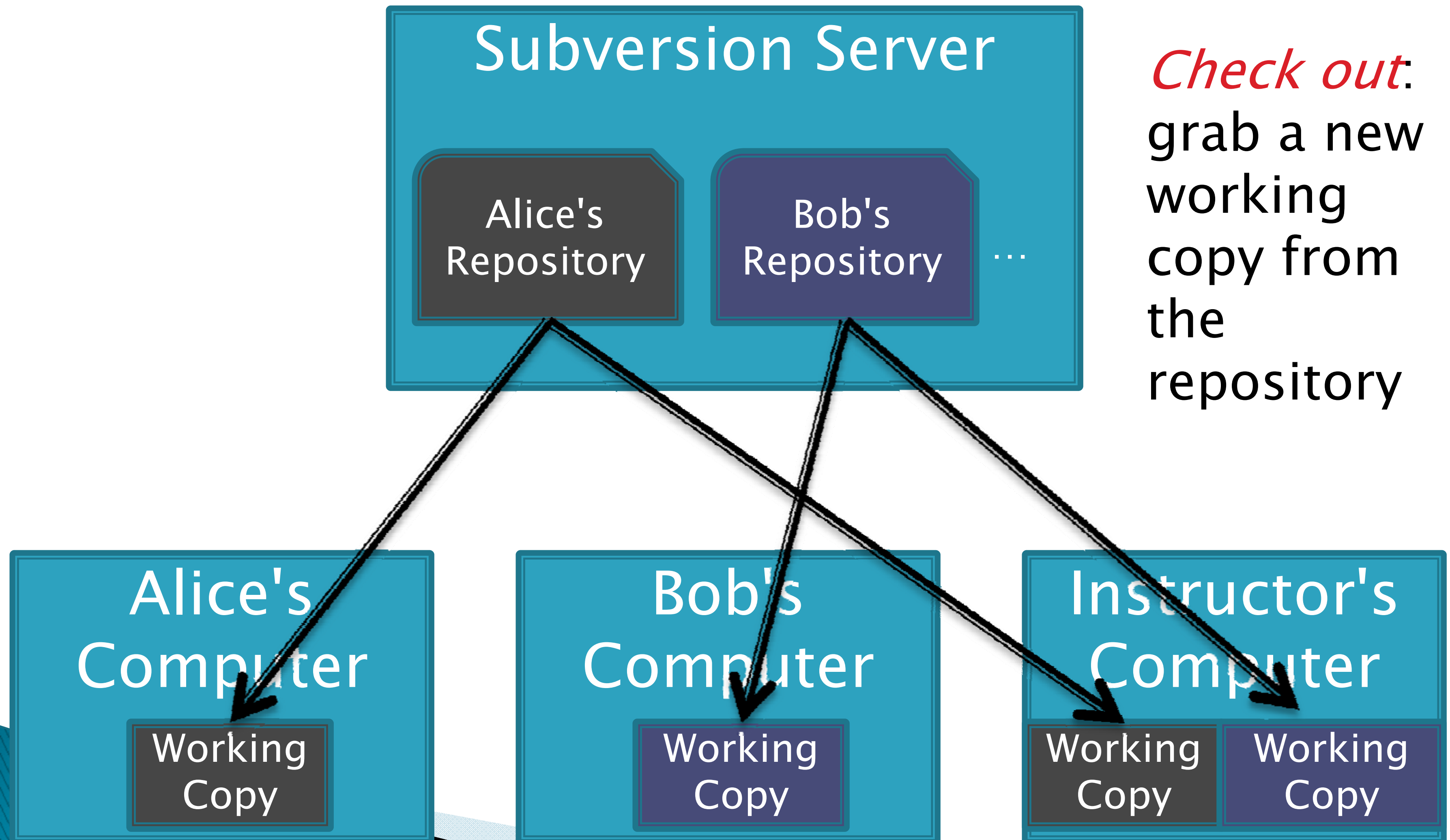


Working copy: the ***current*** version of your data on your computer



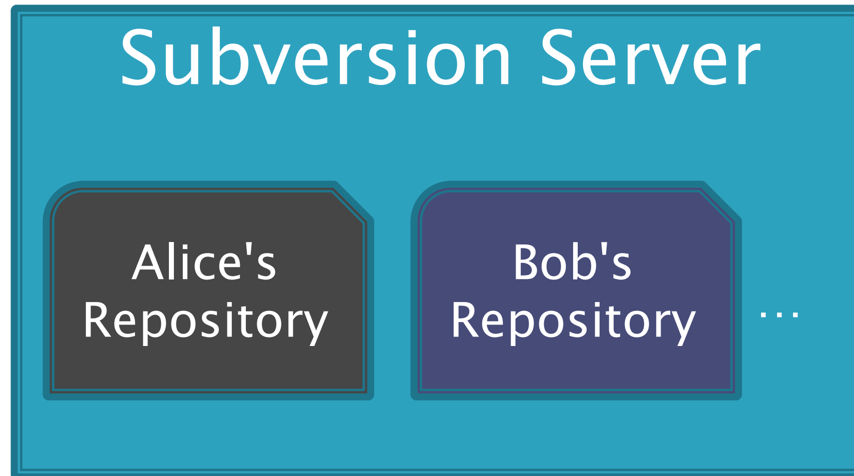


Version Control Steps—Check Out

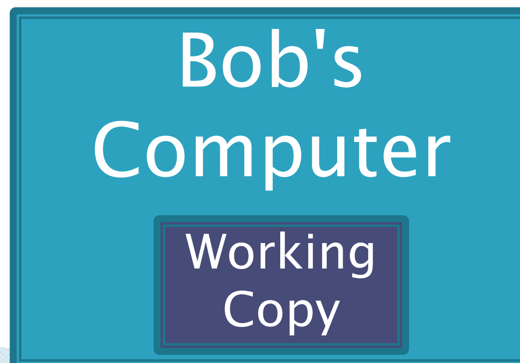


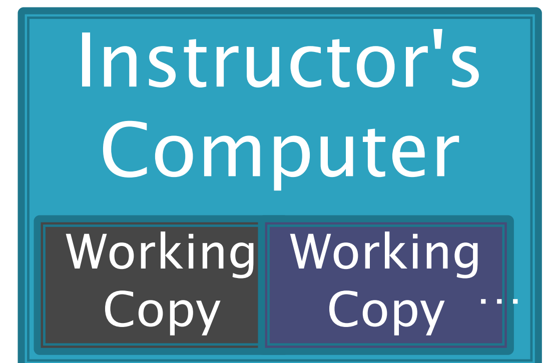
Check out: grab a new working copy from the repository

Version Control Steps—Edit

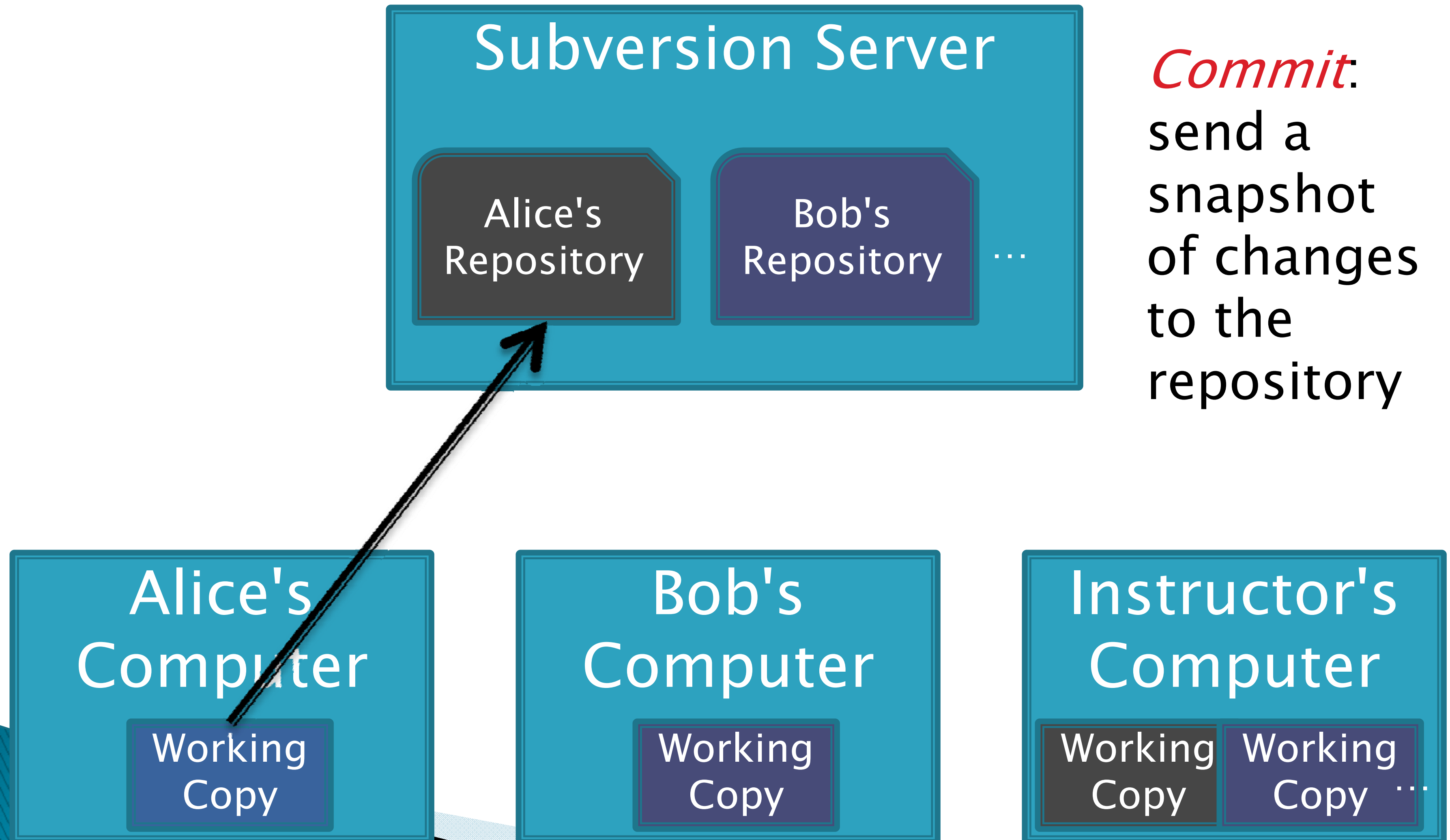


Edit: make ***independent*** changes to a working copy

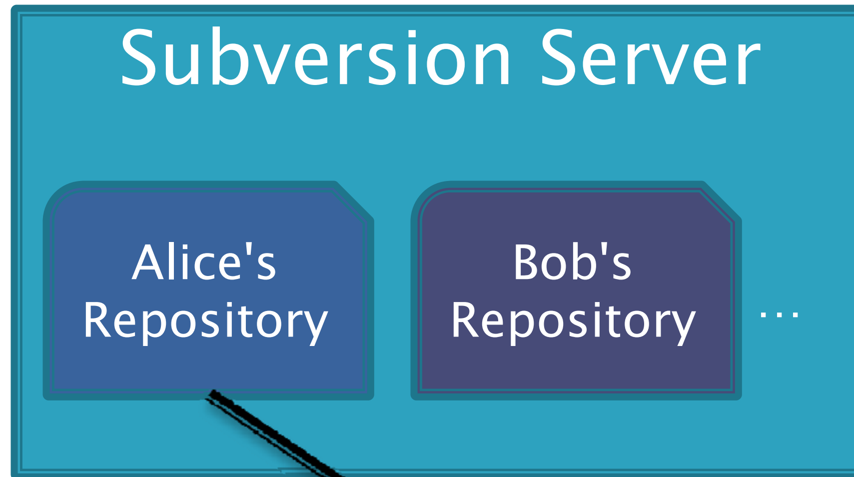


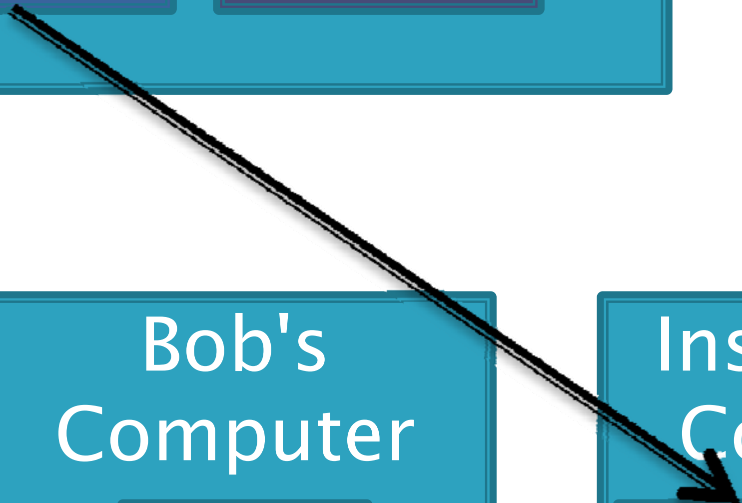


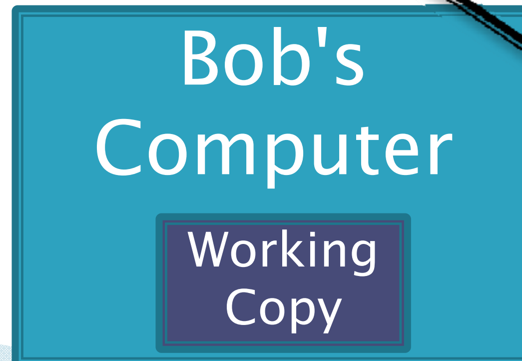
Version Control Steps—Commit

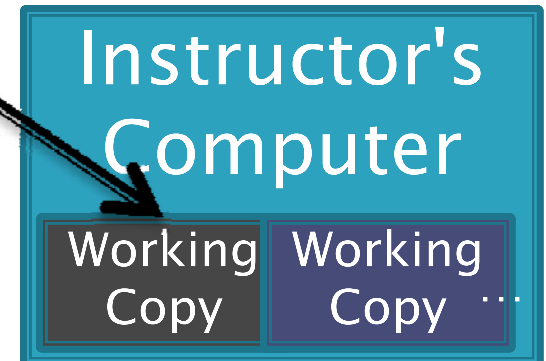


Version Control Steps—Update



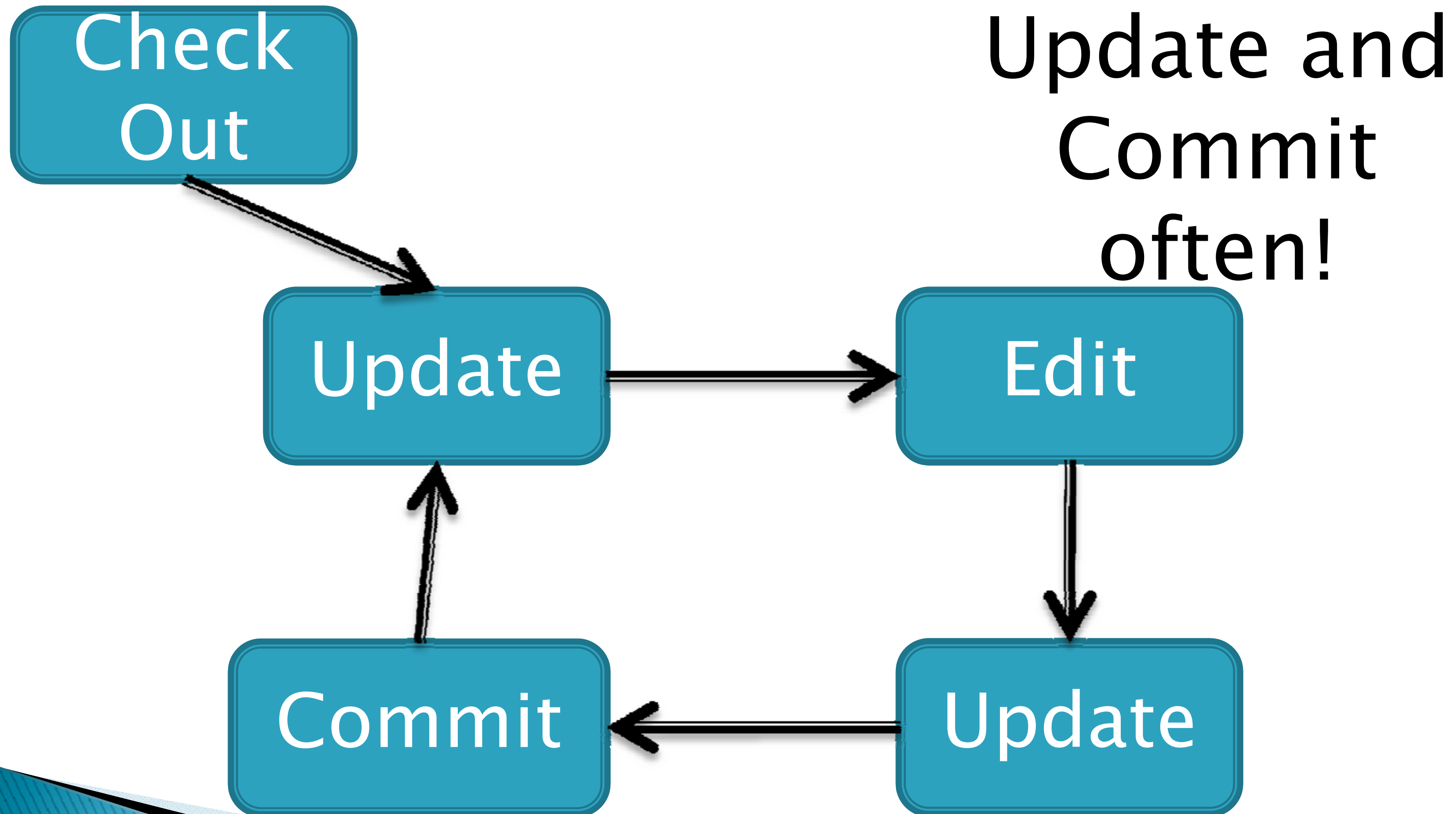




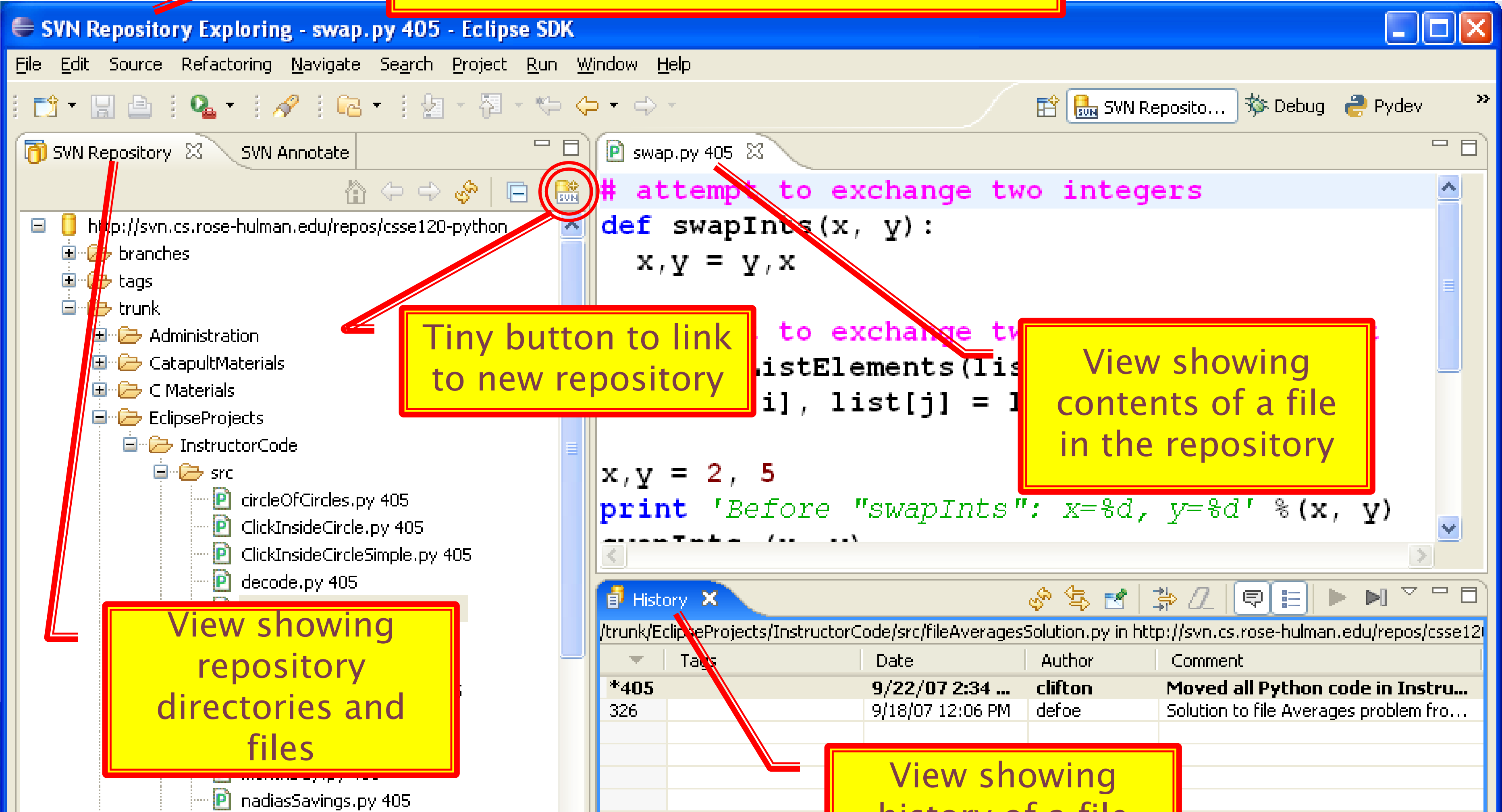


Update: make working copy reflect changes from repository

The Version Control Cycle



Subversion in Eclipse—Subclipse




```
# attempt to exchange two integers
def swapInts(x, y):
  x,y = y,x
x,y = 2, 5
print 'Before "swapInts": x=%d, y=%d' %(x, y)
```

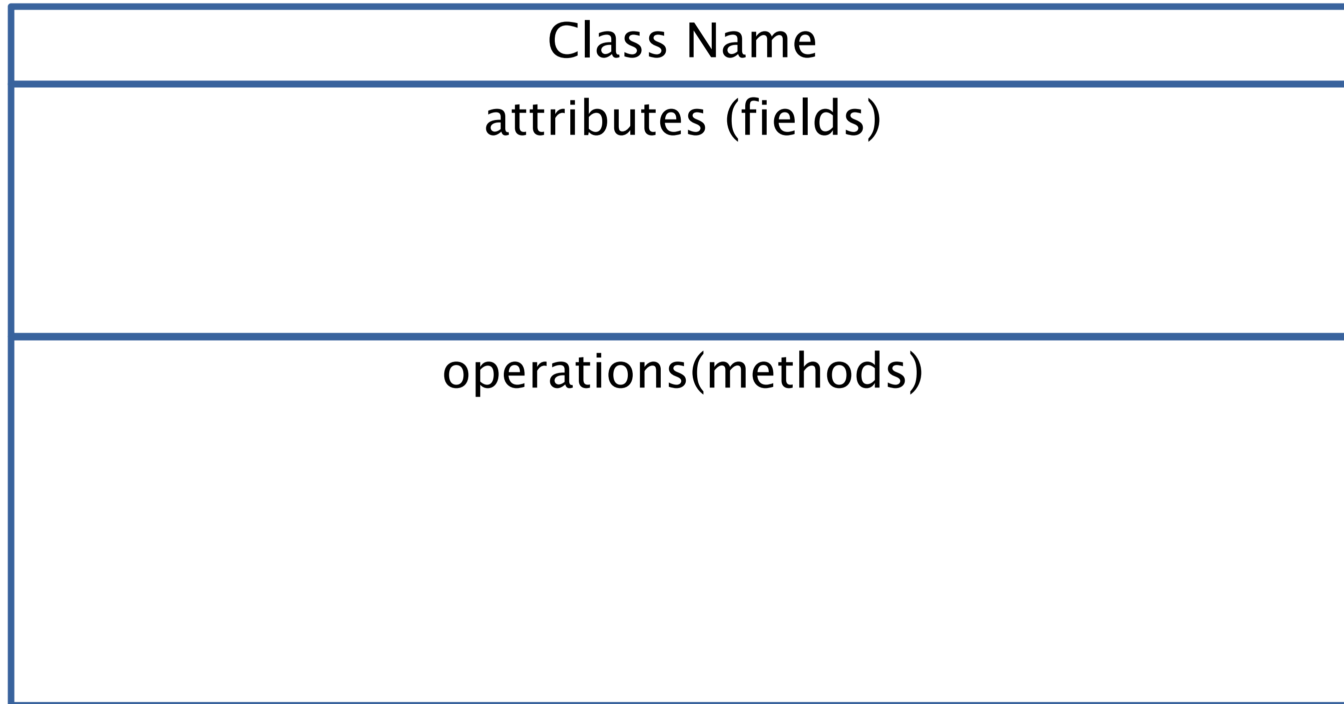
Getting the code for BigRational

- … from your SVN Repository
  - Go to HW5
    Then click the BigRational link
  - Follow the instructions for steps 1–8 of the Check Out the Project section
  - Get help as needed

Yesterday: Objects and Classes

- Hopefully after doing this exercise, you know experientially the meanings of these terms:
  - object
  - class
  - **instance**
  - field
  - method
  - constructor
  - **private** (information hiding)
  - encapsulation
  - **this**

Simple UML Class Diagram



Draw part of the Rectangle diagram

Inheritence

- The Java word for "inherits from" is extends.
- An extension class (subclass) has all of the fields and methods of the extended class (superclass), plus
  - perhaps some new fields
  - and almost always some new or overridden methods.
  - A term that almost always applies to inheritance is "IS-A".
  - **Example**: A square IS-A Rectangle

Other natural examples

- A **Sophomore** IS-A **Student** IS-A **Person.**
- A **Continent** IS-A **LandMass**
- An **HPCompaqNW8440** IS-A **Laptop Computer**
- An **iPod** IS-A **MP3Player**
- A **Square** IS-A **Rectangle**

- It is not true that a **Continent** IS-A **Country** or vice-versa.
- Instead, we say that a **Continent** HAS-A **Country.**

Examples From the Java API Classes

- String extends Object
- ArrayList extends AbstractCollection
- IOException extends Exception
- BigInteger extends Number
- BufferedReader extends Reader
- JButton extends Component
- MouseListener extends EventListener
- Frame extends Window

Extend the Rectangle Class

- Write Square
- Do we need new Instance Variables?
- What methods can/should we override?
- Do it.
- Is this code legal:
  - Rectangle r = new Square(…);
  - Square s = new Rectangle(…);

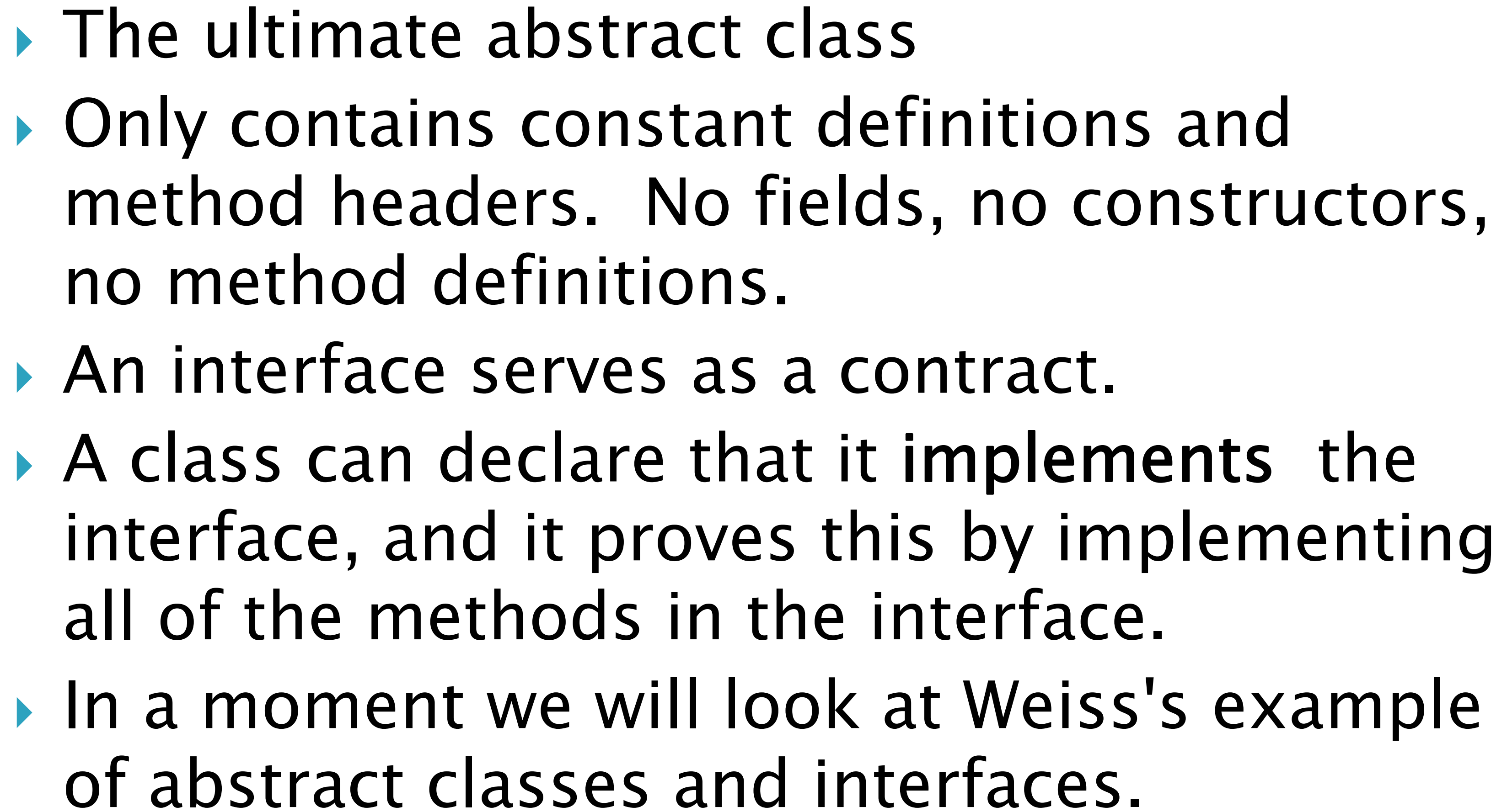
Can we refactor …

- … to find a common ancestor for Circle and Rectangle?
- What is a good name for it?
- What fields/methods can it have?
- We really need an Abstract class. An example soon …

Abstract class

- Not all methods are defined.
- For some we just have stubs.
- Cannot instantiate.

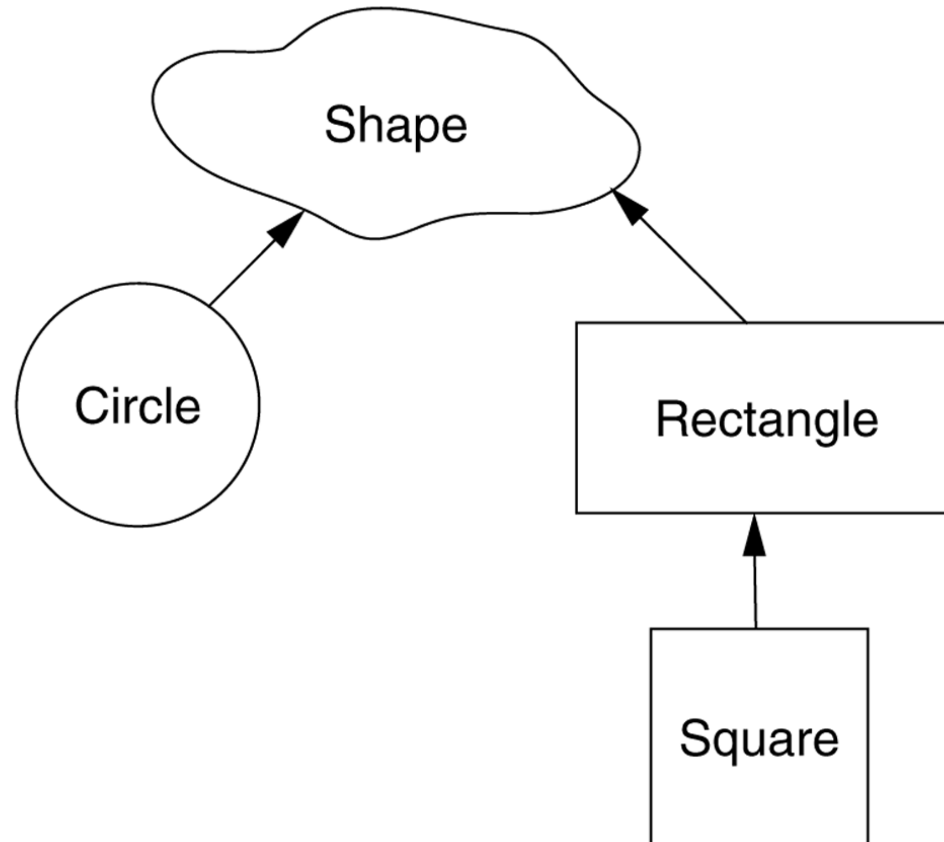
Interface



Shape Hierarchy

Figure 4.10
The hierarchy of shapes used in an inheritance example

Actually, we can (and will) do better, making Shape be an interface.



 © 2002 Addison Wesley

The Shape Interface

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

     public double perimeter();

     public double semiPerimeter();
}
```

AbstractShape class definition

```
public abstract class AbstractShape implements Shape
{
    public abstract double area( );
    public abstract double perimeter( );

    public int compareTo( Object rhs ) {
        double diff = area( ) - ((Shape)rhs).area( );
        if( diff == 0 )
            return 0;
        else if( diff < 0 )
            return -1;
        else
            return 1;
    }

    public double semiPerimeter( ) {
        return perimeter( ) / 2;
    }
}
```

Note that we can use **area** and **perimeter** in the definitions of **compareTo** and **semiperimeter**, even though the former are not implemented in this class.

compareTo is not required to return these specific values. Why does Weiss do it this way?

Circle class definition

```java
public class Circle extends AbstractShape {

    private double radius;

    public Circle( double rad ) {
        radius = rad;
    }

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

    public double perimeter( ) {
        return 2 * Math.PI * radius;
    }

    public String toString( ) {
        return "Circle: " + radius;
    }
}
```

Rectangle class definition

```
public class Rectangle extends AbstractShape {

    private double length;
    private double width;

    public Rectangle( double len, double wid ) {
        length = len; width = wid;
    }

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

    public double perimeter( ) {
        return 2 * ( length + width );
    }

    public String toString( ) {
        return "Rectangle: " + length + " " + width;
    }

    public double getLength( ) {
        return length;
    }

    public double getWidth( ) {
        return width;
    }
}
```

implements the abstract methods

overrides a method from the Object class

Methods unique to this class

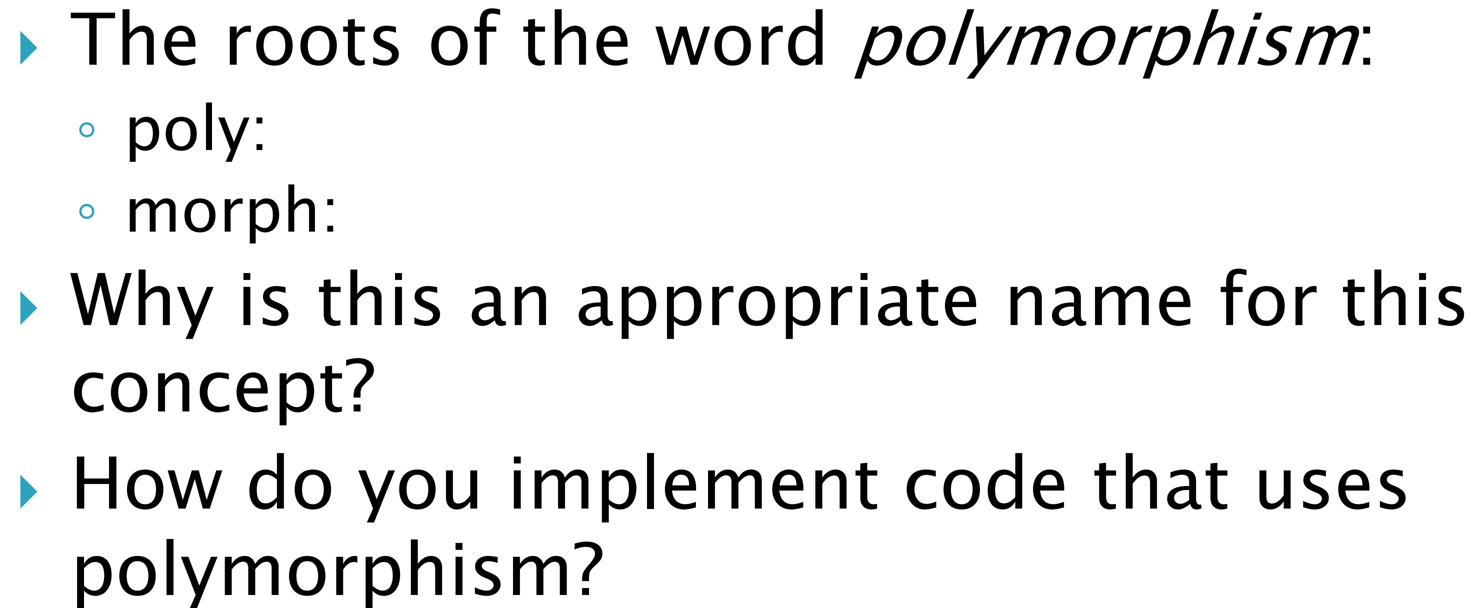
Square class definition

- Square inherits almost all of its functionality from Rectangle.

```
public class Square extends Rectangle  {
    public Square( double side ) {
        super( side, side );
    }

    public String toString( ) {
        return "Square: " + getLength( );
    }
}
```

Polymorphism



- The roots of the word *polymorphism*:
  - poly:
  - morph:
- Why is this an appropriate name for this concept?
- How do you implement code that uses polymorphism?

Polymorphism is possible because of ...

dynamic binding of method calls to actual methods.

The class of the actual object is used to determine which class's method to use.

We'll see it in the ShapesDemo code.

Shape demo part 1

```
class ShapeDemo {
    public static double totalArea( Shape [ ] arr ) {
        double total = 0;
        for( int i = 0; i < arr.length; i++ ) {
            if( arr[ i ] != null )
                total += arr[ i ].area( );
        }
        return total;
    }

    public static double totalSemiperimeter( Shape [ ] arr ) {
        double total = 0;
        for( int i = 0; i < arr.length; i++ ) {
            if( arr[ i ] != null )
                total += arr[ i ].semiPerimeter( );
        }
        return total;
    }
```

If we don't test for null, we could get a NullPointerException.

How do we see polymorphism in action here?

Why are these methods static?

To do before Session 6