Lecture 10 Review of classes

hc293. I tried replying to your email, but you were over your quota and it bounced.

Aaron Macy. I tried replying to your email, but the reply address was here@cornell.edu, which does not exist. How did you do that? I could not find you in the Cornell electronic directory. See me before/after class.

The only inalienable ~~right~~ ~~responsibility~~ pain-in-the-butt of Americans is to receive email. anonymous

God answers knee-mail. anonymous

Assignment A3 handed out today

Due next Friday.

Class NoSuchElementException is defined in package java.util.

Do **not** put a throws clause in your method headers unless the program won't compile without it. You cannot put one in a method that is required by an implements-clause.

To make an apptmnt for a one-on-one:

With Gries: Call Cindy Pakkala at 255-8240

With a TA: See them at recitation or contact them --see website for details.

With a consultant. Go to Upson 304 or Purcell during office hours. If you can't take it then, sign up on the sign-up sheet.

Review

See the Java bootcamp slides.

Look at the beginning of Weiss.

Read ProgramLive, chaps. 1, 3, 4.

Listen to lectures on ProgramLive.

Classes

Assume you know the basics of a class definition. Think of a class as a file-drawer, and the class definition gives the format of all the manilla folders (objects) in the file-drawer. So, a class definition is a template.

All non-static components (variables, methods) go in each folder.

All static components go directly in the file-drawer, along with all the folders. There is only one copy of each of the static components.

Class

```
public class C {
   private int x;
   private double y;
   public void m1(int) { …}
   public int m2() { … }
}
```

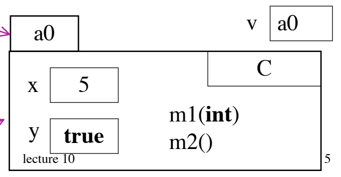
public components can be referenced anywhere; private ones only in the class itself.

Generally, but not always, fields are private and methods are public.

v contains a pointer to, reference to, or the name of the object.

Tab contains name of object (address in memory)

Object drawn like a manilla folder



Summary of classes

- Class defines content of file drawer and format of objects:
- File drawer contains static components and created objects, drawn as manilla folders. The name of an object —its location in memory— is drawn on the tab of the folder.
- new-expression, used to create objects. Know the 3 steps in evaluating it.
- Constructor: called in new-expression to initialize fields.
- Use of **private** and **public.**
- Getter and setter methods.
- static vs. non-static variables (instance variables or fields).
- static vs. non-static methods (instance methods).
- Method toString.
- Two uses of keyword **this**.

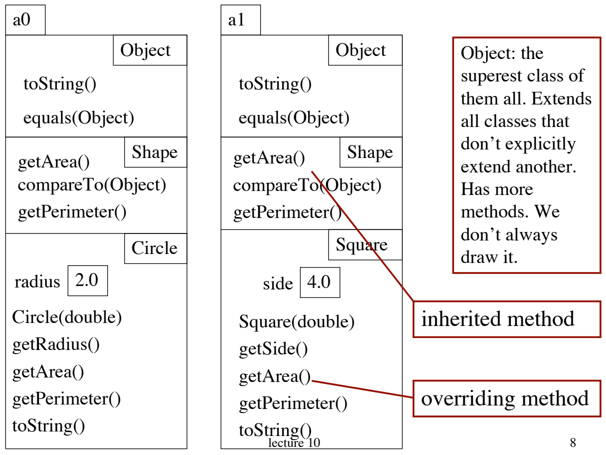
use of this

```
public class C{
  int  x; int y;
  // Constructor: …
  public C (int x, int y;) {
     this.x= x;
     this.y= y
  }
  // Constructor: …
  public C(int y) {
     this(0, y);
     y= y+1;
  }
}
```

when used in this fashion, **this** refers to the object in which method C occurs. So, **this**.x refers to component x of the object.

when used in this fashion, you have a constructor call on a constructor in this class. Must be first statement in constructor.

Subclasses



Subclasses

```
public abstract class Shape
       implements Comparable {
   /** = …    */
   public int compareTo(Object ob)
      { …   }

   // = the area of the shape
   public abstract double getArea();

   // = the circumference of the shape
   public abstract double getPerim();
}
```

```
public class Square extends Shape {
   private double side;

   /** Constructor: …/
   public Square(double s)
     { side= s; }

   public double getSide()
      {  return side;  }

   public double getArea()
      { return side * side; }

   public double getPerimeter()
      { return 4 * side; }

   public String toString()
      { return "Sq: " + side; }
}
```

is-a relation

When designing a set of classes, use the **is-a** relation to help you decide about subclasses.

If a square is a shape, then make Square a Subclass of Shape.

The subclass has more properties than the superclass.

```
public abstract class Shape
       implements Comparable {
   /** = …    */
   public int compareTo(Object ob)
      { …   }

   …
}
```

```
public class Square extends Shape {
   private double side;

   /** Constructor: …/
   public Square(double s)
     { side= s; }

   …
}
```

Two uses of super

```
public class S {
   /** = …    */
   private int s;

   // Constructor: …
   public C(int s) {
         this.s= s;
   }

   public int m(int p) {
      …
   }
}
```

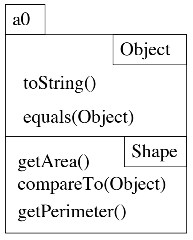
```
public class T extends S {
   private double t;

   /** Constructor: …/
   public SS(int s, double t) {
      super(s);
      this.t= t;
   }

   public int m(int p) {
      …
      int x= super.m(p);
   }
}
```

When used in this fashion, **super** refers to the superclass partition of the object in which method m occurs. Here, **super**.m(p) calls this method.

When used in this fashion, you have a constructor call on a constructor in the superclass. Must be first statement in constructor.

Apparent type: type with which a variable is declared. It determines what can be referenced. **Syntactic property**.

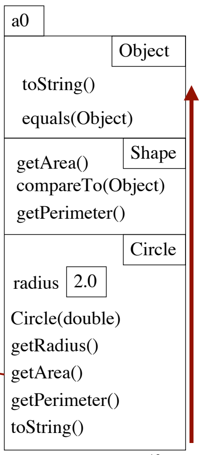
```
public class CompDemo {
   public static void main(String[] pars) {
      Shape[] sh= new Shape[8];
      sh[0]= new Circle(0);
      sh[4]= new Square(0);
      …
   }
}
```



Apparent type of s[0]: Shape.

Only these are legal: sh[0].toString(), sh[0].equals(…), sh[0].getArea(), sh[0].getPerimeter(), sh[0].compareTo(…).

Real type: type of object that is in the variable. It determines is actually referenced. **Semantic property**.

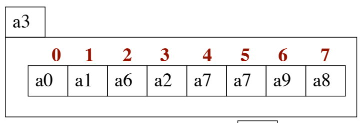


```
public class CompDemo {
   public static void main(String[] pars)
{
      Shape[] sh= new Shape[8];
      sh[0]= new Circle(0);
      sh[4]= new Square(0);
      …
   }
}
```

sh[0].getArea() calls this

sh[0] $\boxed{\text{a0}}$

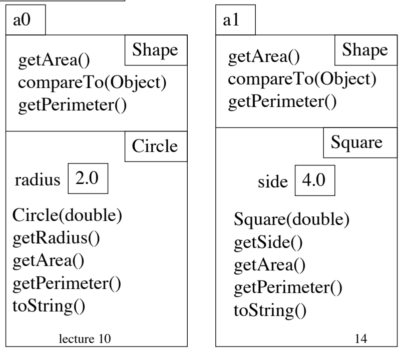
Arrays are objects



b $\boxed{\text{a3}}$

Shape[] b;
b= new Shape[8];
b[0]= new Circle(2);
b[1]= new Square(4);

b[0].compareTo(b[1])


a0
Shape
getArea()
compareTo(Object)
getPerimeter()
Circle
radius
2.0
Circle(double)
getRadius()
getArea()
getPerimeter()
toString()
a1
Shape
getArea()
compareTo(Object)
getPerimeter()
Square
side
4.0
Square(double)
getSide()
getArea()
getPerimeter()
toString()