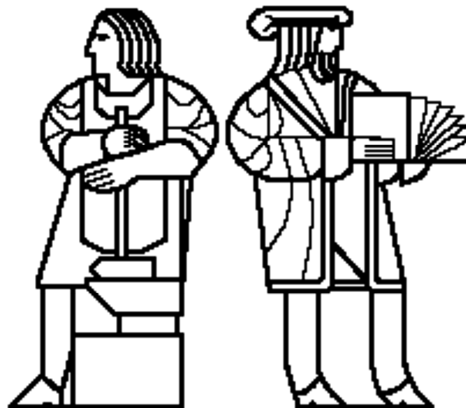
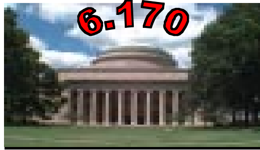
6.170 Lecture 10
Subtyping and subclassing



Michael Ernst
MIT EECS



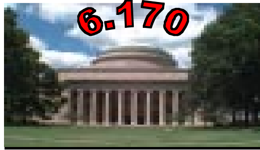
Outline

Subtype polymorphism

Theory of subtypes

Subclassing

- What is subclassing
- Subclassing vs. subtyping
- Potential dangers of subclassing



What is subtyping?

Sometimes every A is a B

library database: every book and CD is a library holding

Subtyping expresses this in the program

programmer declares A is a subtype of B

meaning: "every object that satisfies interface A
also satisfies interface B"

Goal: code written using B's specification operates correctly even if given an A

What is an interface?

interface:

a group of method specifications

a contract at the boundary between modules

using program	↔	ADT implementation
application	↔	operating system
ATM	↔	bank computer

satisfy an interface

provide an implementation that satisfies each of the specifications

Java terminology

```
interface I extends J { }
```

I and J contain signatures only, no implementations
I is a subtype of J

```
class A implements I { }
```

objects of class A satisfy interface I
A is an implementation of I

```
class A extends B { }
```

objects of class A reuse the implementation of B
A is a subclass of B
type of A is also a subtype of type of B

```
abstract class C { }
```

C has unimplemented signatures

Subtyping example

```
interface GObj {                    // this is the SUPER type
   void draw();
   // effects:  paint this on the screen
   Rectangle boundingBox();
   // returns:  the region within which
   //           this is painted when draw() is called.
 }

interface ColorObj extends GObj {   // SUB type
   void setColor(Color c);// effects: this.color = c
   Color getColor();      // returns: this.color
 }

interface TextField extends GObj {
   String getText();
   // returns: the text currently stored in this
 }
```

Java details of subtyping

ColorObj includes all GObj methods

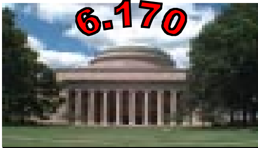
```
interface ColorObj extends GObj {
  void setColor(Color c);
  Color getColor();
  // java adds these lines if you don't write them
  void Draw();
  Rectangle boundingBox();
}
```

All ColorObjs may be assigned to GObj variables

```
ColorObj co = new ColorObj1();  // concrete subtype
GObj g = co;        // Always OK
```

If a Gobj variable refers to a ColorObj, can cast it

```
ColorObj co2 = (ColorObj) g;
  // throws ClassCastException if not a ColorObj
```



Polymorphism: avoids type case

```
Vector objsOnScreen;
void drawAll() {
   Iterator i = objsOnScreen.iterator();
   while (i.hasNext()) {
      GObj g = (GObj) i.next();
      g.draw();
   }}
void drawAll() {
   Iterator i = objsOnScreen.iterator();
   while (i.hasNext()) {
      Object o = i.next();
      if (o instanceof ColorObj) {
         ColorObj c1 = (ColorObj) o;
         c1.draw();
      } else if (o instanceof TextField) {
         TextField t1 = (TextField) o;
         t1.draw();
      }
   }}
```

Subtype polymorphism benefits

Benefits to code that uses GObj:

code reuse

code applies without change to ColorObj, TextField

new subtypes can be added without change

reduced complexity

programmer need only understand GObj interface, not ColorObj, TextField details

better modularity

no dependency if ColorObj or TextField changes

Theory of subtyping

Does A extends B always imply A is a subtype of B?

Example:

```
interface ColorObj extends GObj {
  ...
  void Draw();
  // requires: the color of this has been set
  // effects:  paint this on the screen
}

void drawAll() {
  Iterator i = objsOnScreen.iterator();
  while (i.hasNext()) {
     GObj g = (GObj) i.next();
     g.draw();
  }
}
```

Effects of restricting ColorObj.draw()

Response 1: no change to application

Application doesn't set color in some ColorObj

drawAll() fails

BAD

Response 2: set color before storing in objsOnScreen

rationale for this design not visible in code

can forget the hidden dependency when modifying

BAD

Problem:

GObj.draw() specification not sufficient to use ColorObj.draw() safely

ColorObj is not a GObj

Your turn

Elementary school: every square is a rectangle

```
interface RectangleObj extends GObj {
  void setSize(int w, int h) { ... }
  // effects: this_post.width = w,
  //          this_post.height = h
}
interface SquareObj extends RectangleObj { ... }
```

Choose the best option for SquareObj.setSize

```
1. void setSize(int w, int h);
       // requires: w = h
2. void setSize(int edgeLength);
3. void setSize(int w, int h) throws BadSizeException
       // throws BadSizeException if w != h
```