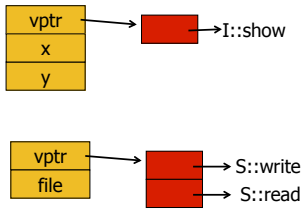
Multiple Inheritance & Java

CSCI 334
Stephen Freund

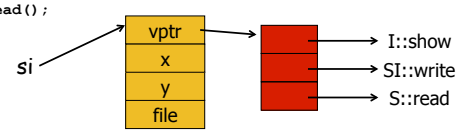
Multiple Inheritance

```
class Image {
  int x,y;
  virtual void show();
};

class Serializable {
  string file;
  virtual void write();
  virtual void read();
};

class SerialImage: public Image, public Serializable {
  virtual void write();
};
```



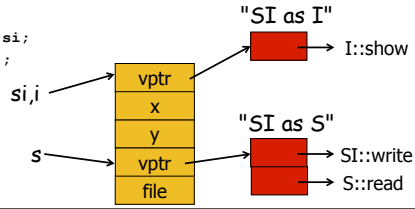
Multiple Inheritance

```
SerialImage *si = new SerialImage();
si -> show();
si -> write();
si -> read();

Image *i = si;
i -> show();

Serial *s = si;
s -> write();
s -> read();
```



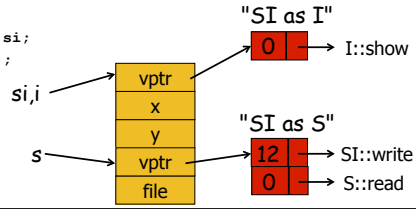
Multiple Inheritance (Not Good Yet...)

```
SerialImage *si = new SerialImage();
si -> show();
si -> write();
si -> read();

Image *i = si;
i -> show();

Serial *s = si;
s -> write();
s -> read();
```



Multiple Inheritance

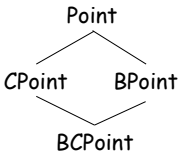
```
SerialImage *si = new SerialImage();
si -> show();
si -> write();
si -> read();

Image *i = si;
i -> show();

Serial *s = si;
s -> write();
s -> read();
```



Name Clashes

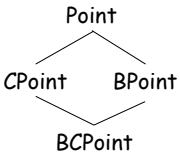
```
class A {
  virtual void f();
};

class B {
  virtual void f();
};

class C : public A, B { }

C *c = new C();
c -> f();
```

Diamond Pattern

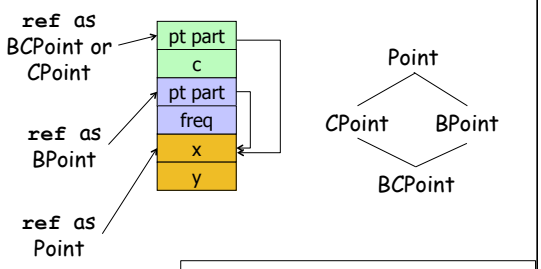



```
class Point {
  int x,y;
}
class ColorPoint : public Point {
  int color;
}
class BlinkingPoint : public Point {
  int freq;
}
class BlinkingColorCPoint :
        public ColorPoint, BlinkingPoint {
  ...
}
```

Diamond Pattern




```
class Point {
  int x,y;
}
class ColorPoint : public virtual Point {
  int color;
}
class BlinkingPoint : public virtual Point {
  int freq;
}
class BlinkingColorCPoint :
        public ColorPoint, BlinkingPoint {
  ...
}
```




```
void print(BPoint p) { ... }
...

ref = new BCPoint(...);
print(ref);
```

Gates Saw Java as Real Threat

Publicly, Microsoft chief Bill Gates was nearly dismissive when he talked in 1996 about Sun Microsystems' Java programming language. But in internal company discussions, he wrote to staff members that Java and the threat the cross-platform technology posed to his company's Windows operating systems "scares the hell out of me."

Wired News Report
8:09 a.m. 22.Oct.98.PDT
(material from '98 trial)

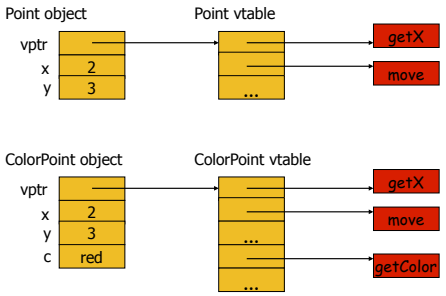
Java Class

```
class Point {
  private int x,y;

  public Point(int xv, int yv) {
    x = xv; y = yv;
  }

  public int getX() { return x; }
  public int getY() { return y; }

  public void move(int dx, int dy) {
    setLocation(x+dx, y+dy);
  }

  protected void setLocation(int xv, int yv) {
    this.x = xv; y = yv;
  }
}
```

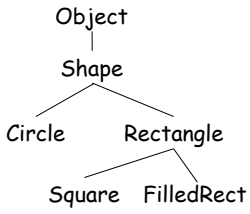
Java Derived Class

```
class ColorPoint extends Point {
  private int c;

  public ColorPoint(int xv, int yv, int cv) {
    super(xv, yv);
    c = cv;
  }

  public int getColor() { return c; }

  public void move(int dx, int dy) {
    super.move(dx, dy);
    darken(1);
  }

  public void darken(int tint) { ... }
}
```

Java Run-Time Representation (more later...)

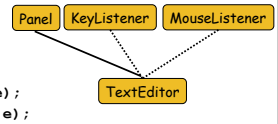


Final Methods




```
class Shape {
  boolean contains(Point p)
}

class Circle extends Shape {
  boolean contains(Point p)
}

class Rectangle extends Shape {
  final boolean contains(Point p)
}
```

Interfaces




```
interface KeyListener {
  void keyPressed(KeyEvent e);
  void keyReleased(KeyEvent e);
  void keyTyped(KeyEvent e);
}

interface MouseListener {
  void buttonClicked(MouseEvent e);
}

class TextEditor extends Panel
                 implements KeyListener,MouseListener {
  void keyPressed(KeyEvent e) { /* code */ }
  void keyPressed(KeyEvent e) { /* code */ }
  void keyPressed(KeyEvent e) { /* code */ }
  void buttonClicked(MouseEvent e) { /* code */ }
 }
```

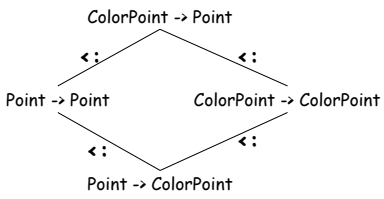
Interfaces vs. Multiple Inheritance

```
class DefaultKeyListener {
  // default is to do nothing
  void keyPressed(KeyEvent e)  { }
  void keyReleased(KeyEvent e) { }
  void keyTyped(KeyEvent e) { }
}

class TextField : public Panel, DefaultKeyListener {
  void keyTyped(KeyEvent e) { /* code here */ }
  ...
}

class TerminalWindow : public Panel, DefaultKeyListener {
  void keyTyped(KeyEvent e) { /* code here */ }
  ...
}
```

Interfaces vs. Multiple Inheritance

```
interface KeyListener {
  void keyPressed(KeyEvent e);
  void keyReleased(KeyEvent e);
  void keyTyped(KeyEvent e);
}

class TextField extends Panel implements KeyListener {
  void keyPressed(KeyEvent e) { }
  void keyTyped(KeyEvent e) { /* code here */ }
  void keyReleased(KeyEvent e) { }
}

class TerminalWindow extends Panel implements KeyListener {
  void keyPressed(KeyEvent e) { }
  void keyTyped(KeyEvent e) { /* code here */ }
  void keyReleased(KeyEvent e) { }
}
```

Function Subtyping Revisited



- Return Types are *Covariant*
- Argument Types are *Contravariant*

Java Array Covariant Subtyping Rule

```
class Point { ... }
class ColorPoint extends Point { ... }

Point pts[] = new Point [100];


pts[0] = new Point (10,10);
pts[1] = new Point (20,20);
```

Java Array Covariant Subtyping Rule

```
class Point { ... }
class ColorPoint extends Point { ... }

ColorPoint cpts = new ColorPoint[100];
Point pts[] = cpts;

pts[0] = new Point (10,10);
pts[1] = new Point (20,20);

 ...
cpts[0].setColor(RED);
```

Why Did They Add It?

```
static void arrayCopy(Point src[], Point dst[]) {
  for (int i = 0; i < src.length; i++)
    dst[i] = src[i];
}


Point p[];
Point q[];
arrayCopy(p,q);

String s[];
String t[];
arrayCopy(s,t);
```

General arrayCopy Operation

```
static void arrayCopy(Object src[], Object dst[]) {
  for (int i = 0; i < src.length; i++)
    dst[i] = src[i];
}


Point p[];
Point q[];
arrayCopy(p,q);

String s[];
String t[];
arrayCopy(s,t);
```

From Bill Joy (Sun Cofounder)

Date: Fri, 09 Oct 1998 09:41:05 -0600
From: bill joy
Subject: …[discussion about java genericity]

actually, java array covariance was done for less noble reasons …:
it made some generic "bcopy" (memory copy) and like operations much easier to write…
I proposed to take this out in 95, but it was too late (…).
i think it is unfortunate that it wasn't taken out…
it would have made adding genericity later much cleaner, and [array covariance] doesn't pay for its complexity today.
wnj

Variance in Scala

```
Array[ColorPoint] <: Array[Point] ?
```

```
PartialFunction[Point,ColorPoint]
                <:
  PartialFunction[Point,Point] ?
```

Variance Annotations in Scala

- Class defined with covariant parameter T:

```
class List[+T] { ... }
```

- So `List[ColorPoint] <: List[Point]`
- Which of these is type-safe?

```
class List[+T] {
  def f(t : T) = ...
}
```

```
class List[+T] {
  def f() : T = ...
}
```

- Function Types:

```
class PartialFunction[-A,+R] { ... }
```