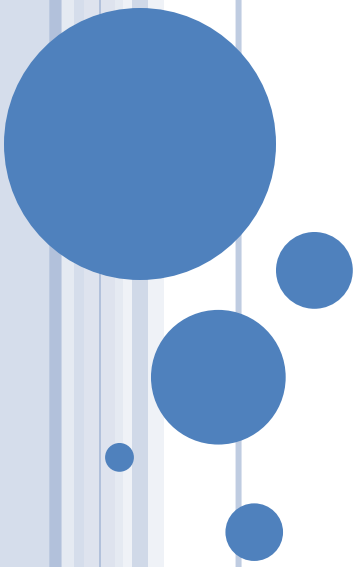


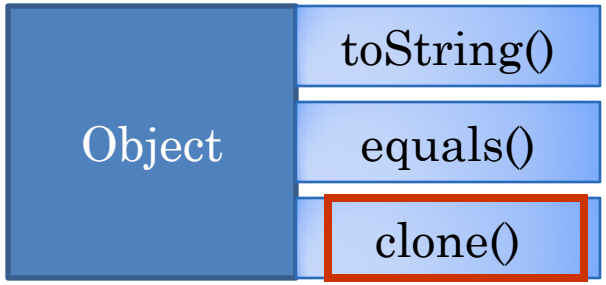
Comp 401
Copy: Shallow and Deep

Instructor: Prasun Dewan

Prerequisite

- Composite Object Shapes
- Inheritance

Clone Semantics?



Need to understand memory representation

Copying Objects

```
p1 = new AMutablePoint(200, 200);
p2 = p1;
p1.setX (100);
```

```
p2.getX() == p1.getX()
```

→ true

- What if we want copy rather than reference.
- The properties can be changed independently
- Backup

Copier does the work

```
p1 = new AMutablePoint(200, 200);
p2 = new AMutablePoint (p1.getX(), p1.getY());
p1.setX (100);
```

p2.getX() == p1.getX()

→ false

COPIED OBJECT DOES THE WORK

```
// in Object, subtype can increase access of overridden method
protected Object clone(){ … }
```

```
public interface CloneablePoint extends MutablePoint {
  public Object clone();
}
```

```
// Cloneable is an empty interface, should be an annotation
public class ACloneablePoint extends AMutablePoint
                        implements CloneablePoint, Cloneable {
  public ACloneablePoint(int theX, int theY) {
    super(theX, theY);
  }
  public Object clone() {
    return super.clone();
  }
}
```

```
CloneablePoint p1 = new ACloneablePoint(200, 200);
CloneablePoint p2 = (CloneablePoint) p1.clone();
p1.setX (100);
```

p2.getX() == p1.getX()

→ false

BoundedPoint Clone

```
public class ACloneableBoundedPoint extends ABoundedPoint
        implements CloneableBoundedPoint, Cloneable {
  public ACloneableBoundedPoint(int initX, int initY,
                CloneablePoint theUpperLeftCorner,
                CloneablePoint theLowerRightCorner) {
    super(initX, initY, theUpperLeftCorner, theLowerRightCorner);
  }
  public Object clone() {
    return super.clone();
  }
}
```

BoundedPoint Clone

```
CloneableBoundedPoint p1 =
  new ACloneableBoundedPoint (75, 75,
       new AMutablePoint(50,50), new AMutablePoint(100,100));
CloneableBoundedPoint p2 = (CloneableBoundedPoint) p1.clone();
p1.setX (100);
p1.getUpperLeftCorner().setX(200);
```

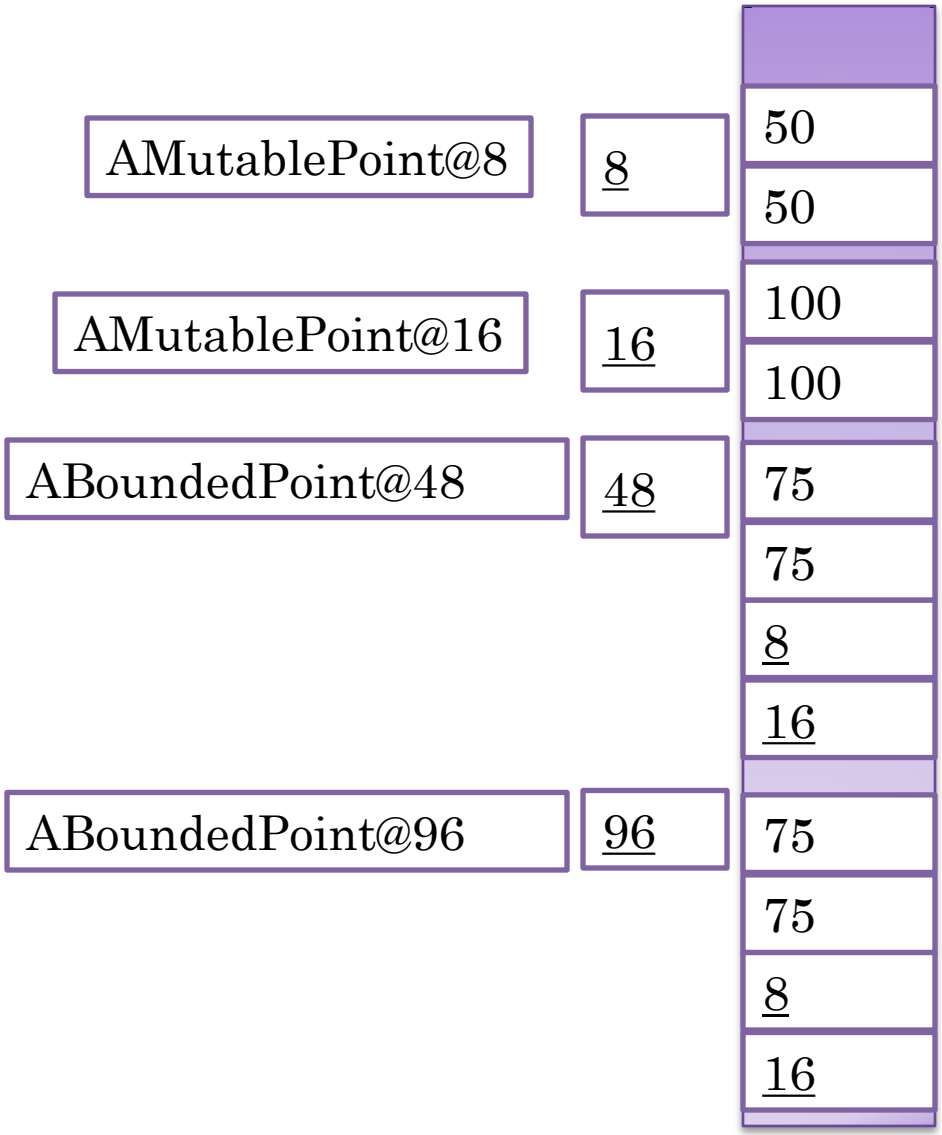
p2.getX() == p1.getX()

→ false

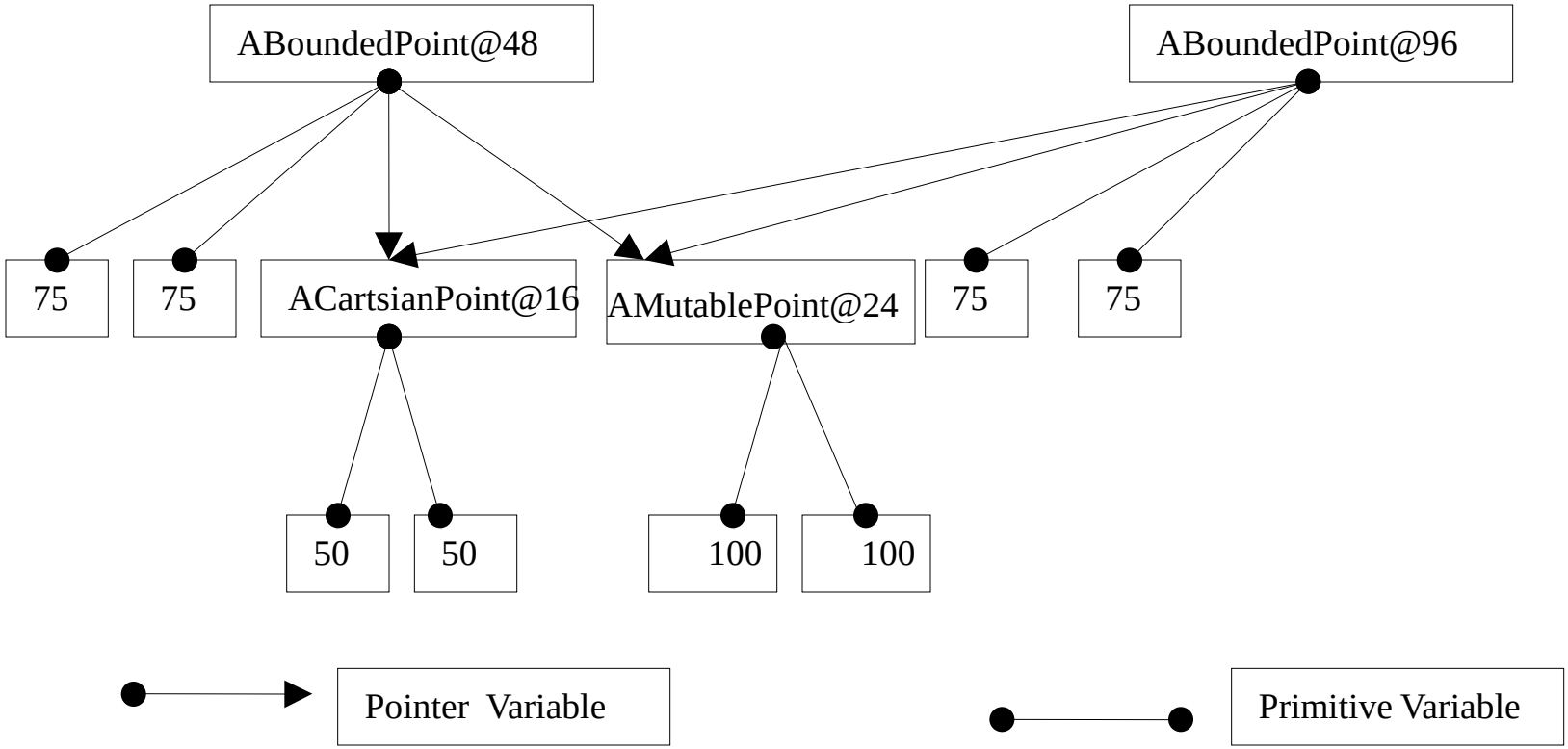
p1.getUpperLeftCorner().getX() ==
p2.getUpperLeftCorner().getX()

→ true

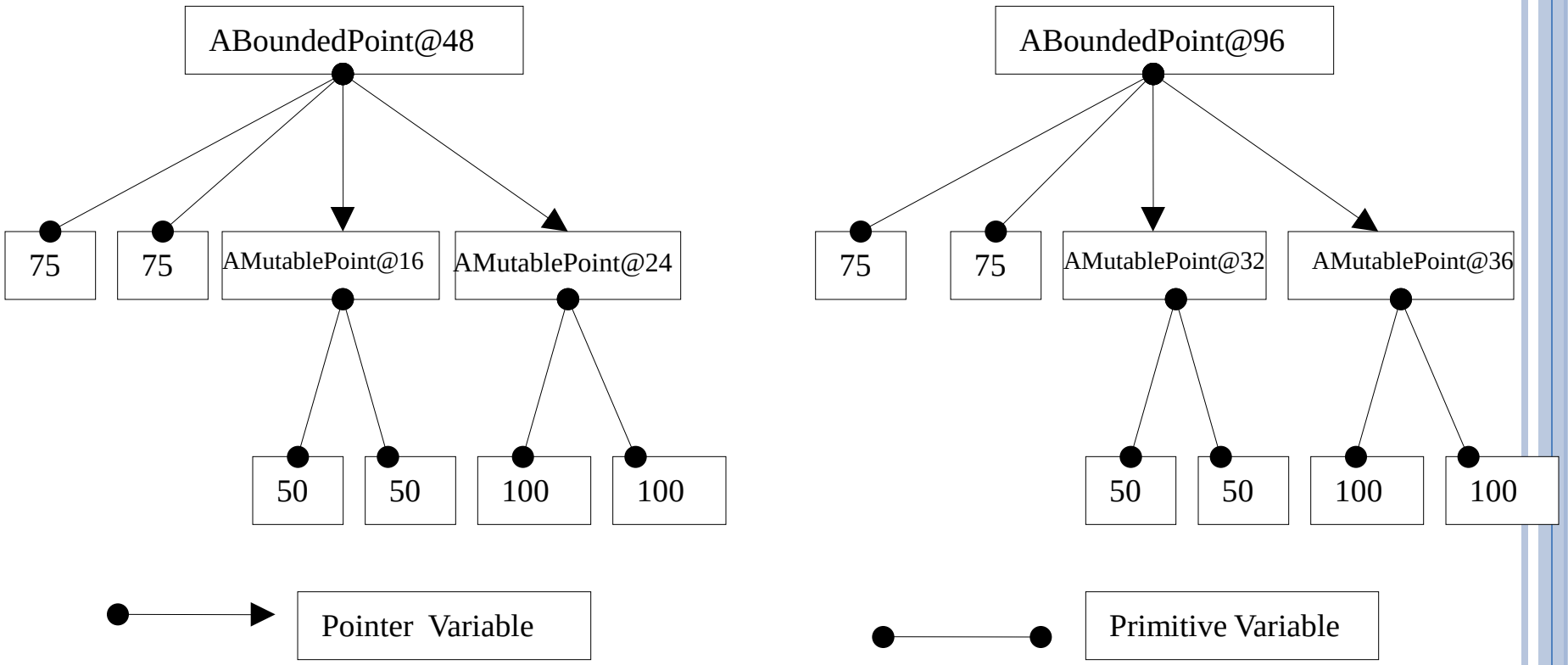
Replicating Instance Variable Values



Shallow Copy



Deep Copy


ABoundedPoint@48
75
75
AMutablePoint@16
AMutablePoint@24
50
50
100
100
ABoundedPoint@96
75
75
AMutablePoint@32
AMutablePoint@36
50
50
100
100
Pointer Variable
Primitive Variable

Object Clone

```
// Object implements shallow copy
protected Object clone() { … }
```

```
//class can implement multiple interfaces, and interface such as
    Cloneable can be empty
public class AMutablePoint implements Point, Cloneable
// Subclass can make it public
public Object clone() { return super.clone() } // need exception
    handling, discussed later
```

Bounded Point Deep Copy ?

```
public Object clone() {


};
```

```
public class ABoundedPoint extends AMutablePoint implements
BoundedPoint {
    Point  upperLeftCorner, lowerRightCorner;
    public ABoundedPoint (int initX, int initY,
                    Point initUpperLeftCorner, Point initLowerRightCorner) {
            super(initX, initY);
            upperLeftCorner = initUpperLeftCorner;
            lowerRightCorner = initLowerRightCorner;
     }
…
}
```

Bounded Point Deep Copy

```
public CloneableBoundedPoint clone() {
  return  new ACloneableBoundedPoint (x, y,
        (CloneablePoint) ((CloneablePoint)upperLeftCorner).clone(),
        (CloneablePoint) ((CloneablePoint)lowerRightCorner).clone());
}
```

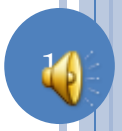
```
CloneableBoundedPoint p1 =
  new ACloneableBoundedPoint (75, 75,
     new ACloneablePoint(50,50), new ACloneablePoint(100,100));
  CloneableBoundedPoint p2 =  p1.clone();
  p1.setX (100);
  p1.getUpperLeftCorner().setX(200);
```

p2.getX() == p1.getX()

→ false

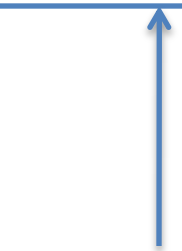
p1.getUpperLeftCorner().getX() ==
p2.getUpperLeftCorner().getX()

→ false



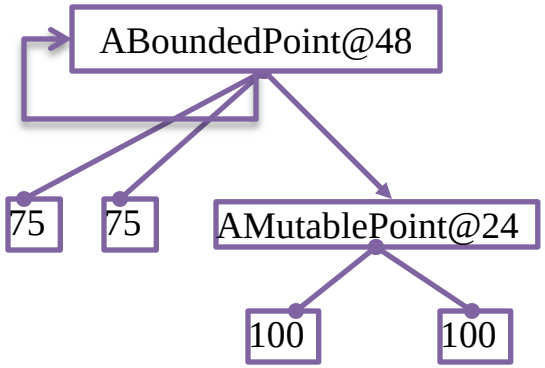
Bounded Point Deep Copy Problems

```
CloneableBoundedPoint p1 =
  new ACloneableBoundedPoint (75, 75,
   new ACloneablePoint(50,50), new ACloneablePoint(100,100));
p1.setUpperLeftCorner(p1);
CloneableBoundedPoint p2 =  p1.clone();
```



Infinite recursion

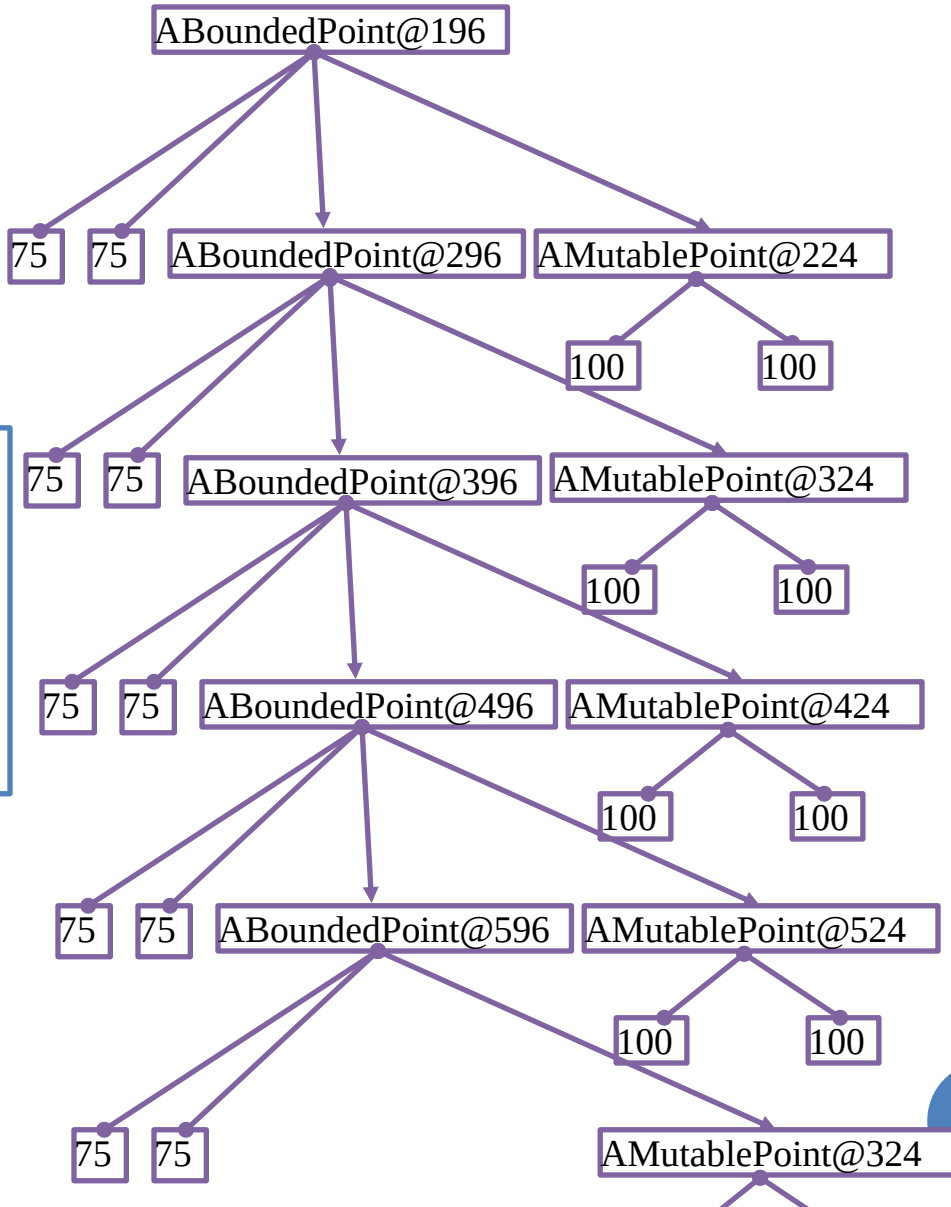
Cloning Graph Structures




```
public CloneableBoundedPoint clone() {
  return
    new ACloneableBoundedPoint (
          x, y,
          upperLeftCorner.clone(),
          lowerRightCorner.clone());
}
```

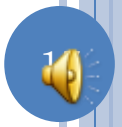
Graph structures are useful and make deep copy problematic

Link from child to parent often occurs



Shallow vs. Deep Copy

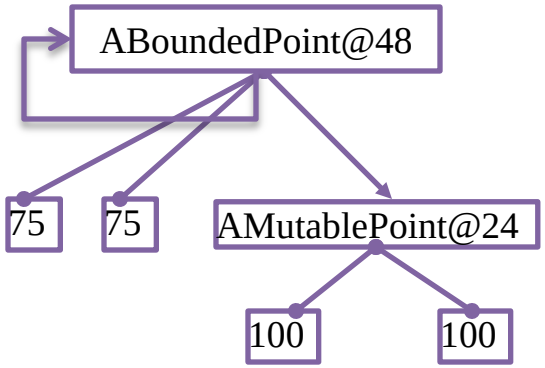
- Shallow copy:
  - Copies the instance but not its components
  - Creates a new object and assigns instance variables of copied object to corresponding instance variables of new object.
- Deep copy
  - Creates a new object and assigns (deep or shallow?) copies of instance variables of copied object to corresponding instance variables of new object.

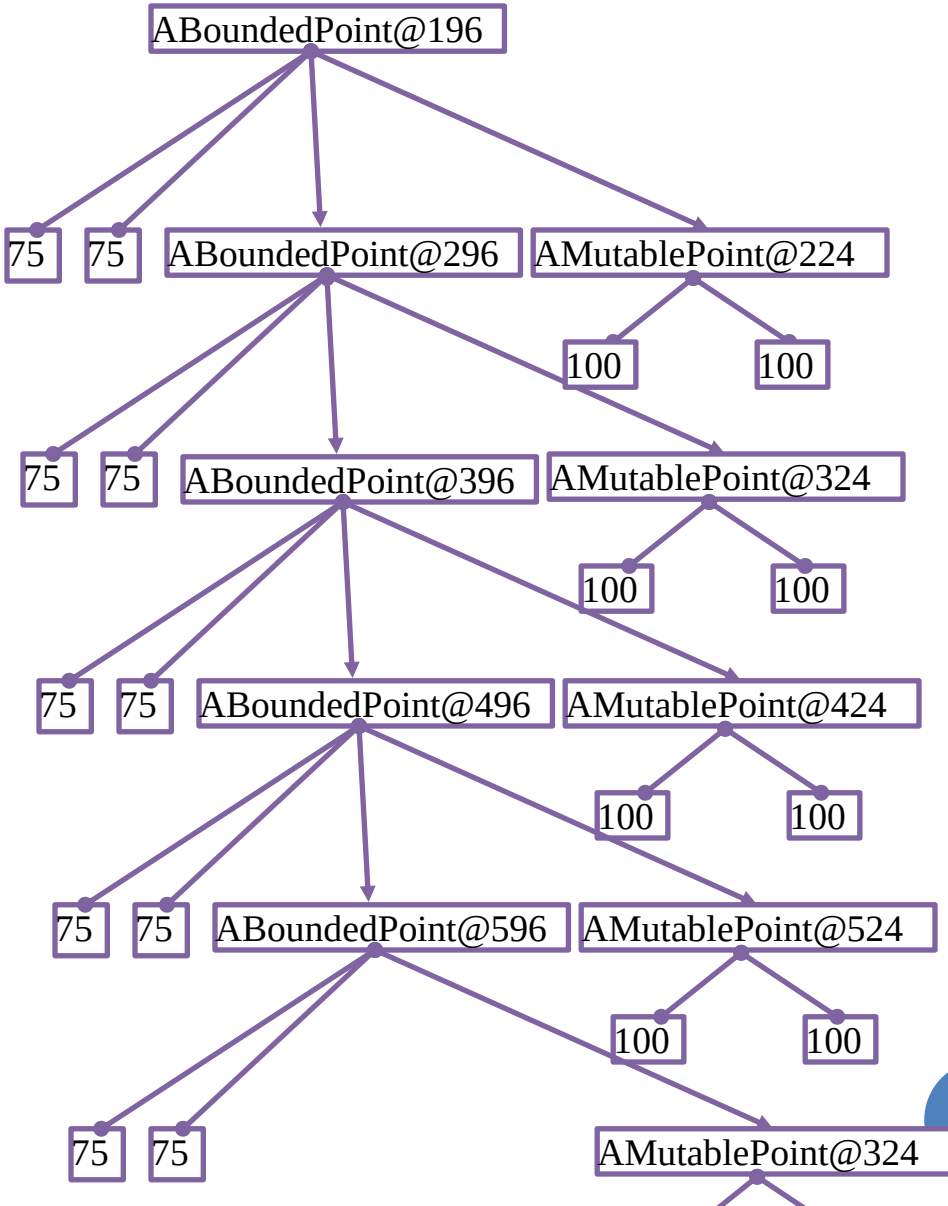


SMALLTALK Shallow, Deep(er), and Regular Copy

- Copy
  - Programmer makes it either shallow or deep copy. By default it is shallow.
- Shallow copy:
  - Copies the instance but not its components
  - Creates a new object and assigns instance variables of copied object to corresponding instance variables of new object.
- Deep copy
  - Creates a new object and assigns copy of each instance variable of copied object to corresponding instance variable of new object.

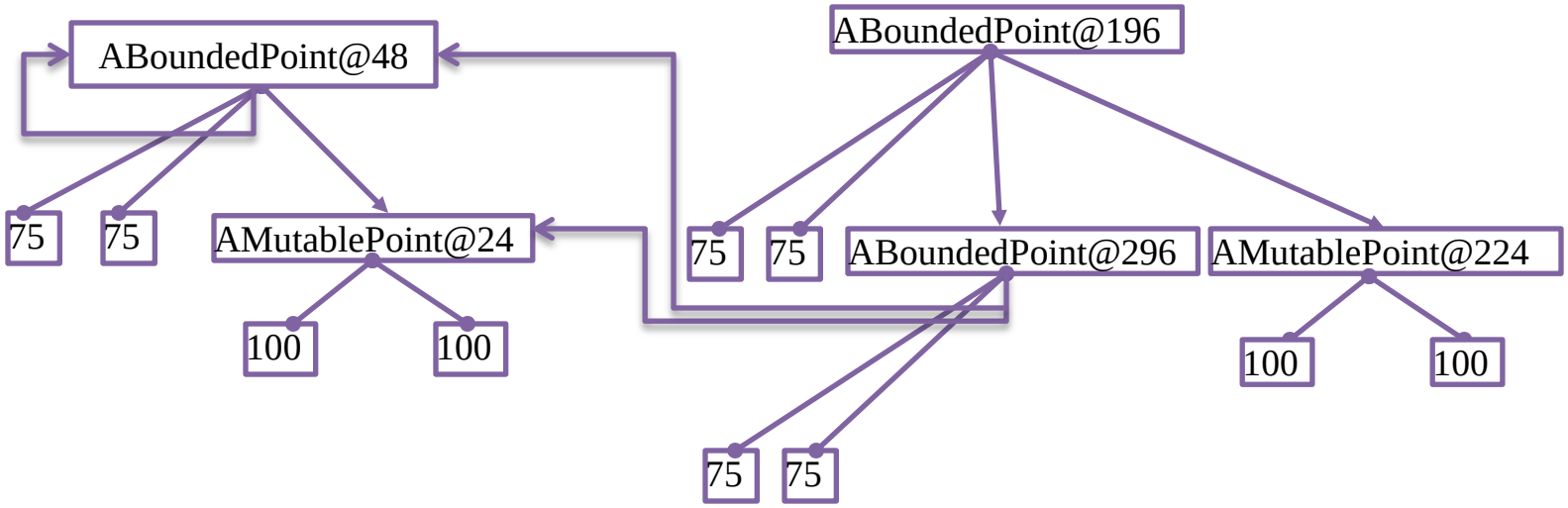
Deep Copy of Graph Structure





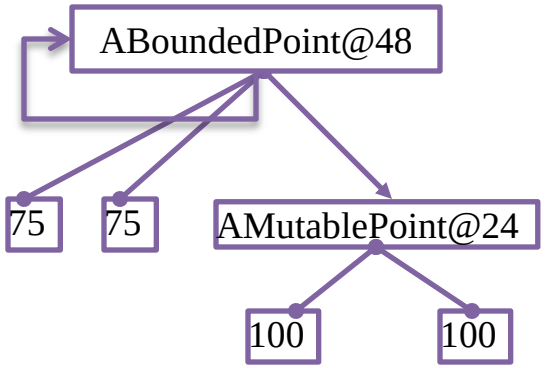
If copy is deepCopy

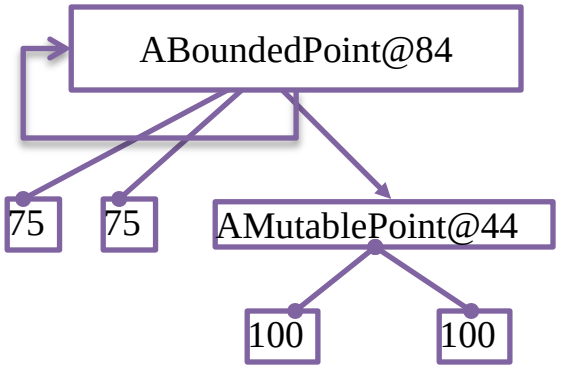
Deep Copy of Graph Structure



If copy is shallowCopy

Isomorphic Deep Copy



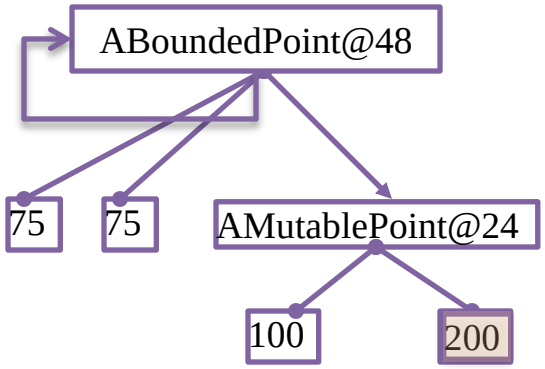


JAVA SERIALIZATION

- Used to copy object (implementing java.io.Serializable empty interface) to file or network
- Deep isomorphic copy
  - Created a deep isomorphic copy of object
- Used by OE library to create a deepCopy
  - **public** Object Misc.deepCopy(Object object)
- Deep copy used for automatic refresh

```
import java.io.Serializable;
public class AMutablePoint extends AMutablePoint implements Point,
Serializable { … }
```

Object Editor Automatic Refreshes

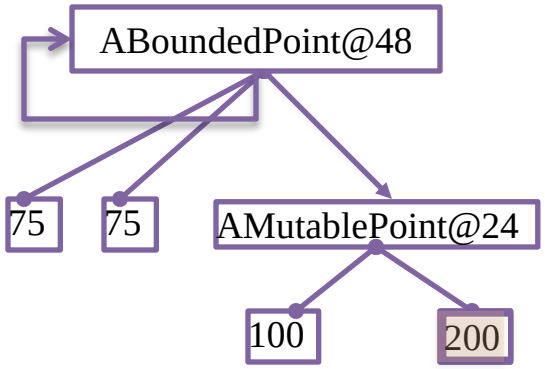


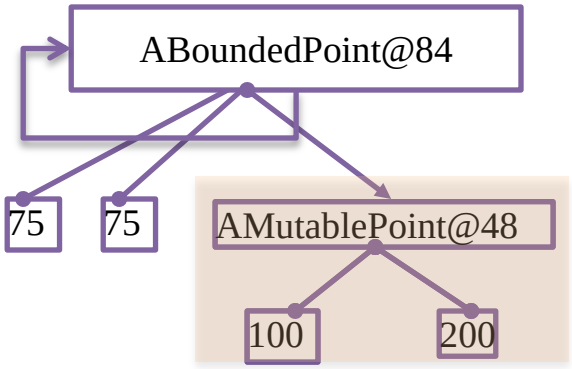
Suppose some operation on ABoundedPoint@48 changes Y coordinate of AMutablePoint@24

How does OE know that the value changed?

All components of ABoundedPoint@48 have the same value

Object Editor Automatic Refreshes





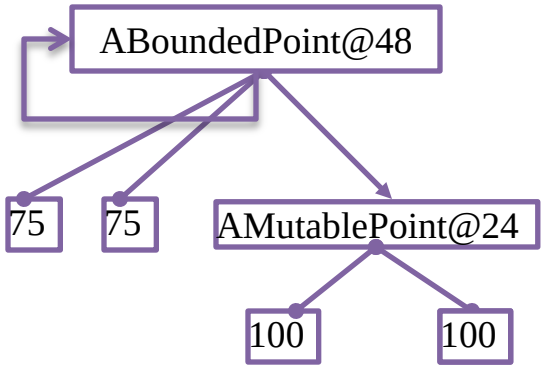
Creates an isomorphic copy when it first encounters an object

Suppose an operation is executed on an object

Calls equals() on new object to compare it with copy

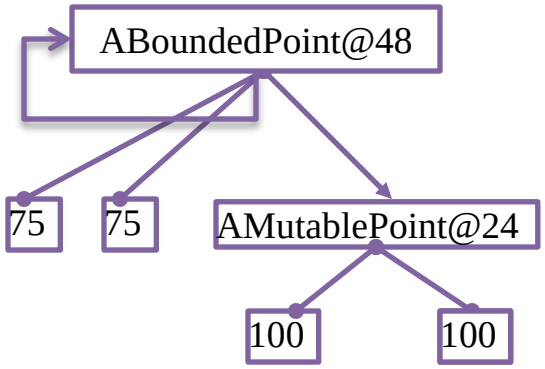
If equals returns false, efficiently updates display of changed object and creates new copy of changed object

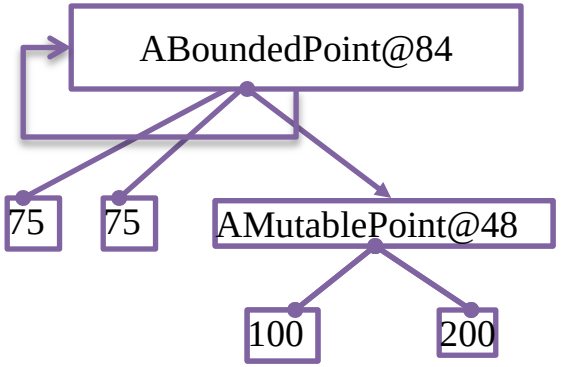
Complete Brute Force Refresh



If object and all of its descendants not Serializable then no deep copy of efficient refresh

Complete Brute Force Refresh





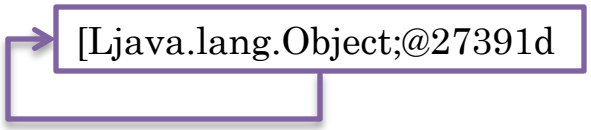
If object and all of its descendants Serializable, but no overridden equals(), then complete brute force refresh

Can override equals() and set break point in it to verify it is called with copy

Why Simplistic Array Print?

[Ljava.lang.Object;@27391d

```
Object[] recursive = new Object[1];
recursive[0] = recursive;
System.out.println(recursive);
```



Infinite recursion if println() recursively printed each element

Other Object Operations

- hashCode()
  - Relevant to hashtables – will learn about in data structures.
  - Think of it as the internal address of object.
- Various versions of wait() and notify()
  - Relevant to threads – will study them in depth in operating systems course
  - See section on synchronization and wait and notify.
  - Useful for animations.
- getClass()
  - Returns the class, on which one can invoke "reflection" methods
  - Used by ObjectEditor to edit arbitrary object.
- finalize()
  - Called when object is garbage collected.

Garbage Collection