COMP 401
COPY: SHALLOW AND DEEP

Instructor: Prasun Dewan
PREREQUISITE

- Composite Object Shapes
- Inheritance
CLONE SEMANTICS?

Object

- toString()
- equals()
- clone()

Need to understand memory representation
COPYING OBJECTS

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

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

```
p2.getX() == p1.getX()  → true
```
Copier Does the Work

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

```
p2.getX() == p1.getX()  \rightarrow  false
```
// in Object, subtype can increase access of overridden method

```java
protected Object clone(){...}
```

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

// Cloneable is an empty interface, should be an annotation
```java
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
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**

```java
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()  \[\rightarrow\] false

p1.getUpperLeftCorner().getX() ==
p2.getUpperLeftCorner().getX()  \[\rightarrow\] true
```
Replicating Instance Variable Values

- AMutablePoint@8: 8 50 50
- AMutablePoint@16: 16 100 100
- ABoundedPoint@48: 48 75 75
- ABoundedPoint@96: 96 75 75
SHALLOW COPY

ABoundedPoint@48

ACartsianPoint@16

AMutablePoint@24

ABoundedPoint@96

75 75

50 50

100 100

Pointer Variable

Primitive Variable
DEEP COPY

ABoundedPoint@48

75 75 AMutablePoint@16 AMutablePoint@24

50 50 100 100

Pointer Variable

ABoundedPoint@96

75 75 AMutablePoint@32 AMutablePoint@36

50 50 100 100

Primitive Variable
// 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
public Object clone() {

};

public class ABoundingPoint extends AMutablePoint implements BoundedPoint {
    Point upperLeftCorner, lowerRightCorner;

    public ABoundingPoint (int initX, int initY,
    Point initUpperLeftCorner, Point initLowerRightCorner) {
        super(initX, initY);
        upperLeftCorner = initUpperLeftCorner;
        lowerRightCorner = initLowerRightCorner;
    }

    ...
}

BOUNDDED 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
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**

Graph structures are useful and make deep copy problematic.

Link from child to parent often occurs.

```java
public CloneableBoundedPoint clone() {
    return new ACloneableBoundedPoint(
        x, y,
        upperLeftCorner.clone(),
        lowerRightCorner.clone());
}
```
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
If copy is shallowCopy
ISOMORPHIC DEEP COPY

ABoundedPoint@48

75 75 AMutablePoint@24

100 100

ABoundedPoint@84

75 75 AMutablePoint@44

100 100
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

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

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

Point p1 = new AMutablePoint(100, 100);

Point p2 = new AMutablePoint(150, 75);

p2 = p1;

Covered elsewhere

AMutablePoint@8
AMutablePoint@76

Garbage Collected

<table>
<thead>
<tr>
<th>address</th>
<th>variables</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>AMutablePoint@8</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>AMutablePoint@8</td>
<td>100</td>
</tr>
<tr>
<td>52</td>
<td>Point p1</td>
<td>8</td>
</tr>
<tr>
<td>56</td>
<td>Point p2</td>
<td>76</td>
</tr>
<tr>
<td>76</td>
<td>AMutablePoint@76</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>AMutablePoint@76</td>
<td>75</td>
</tr>
</tbody>
</table>