CS 211 Accelerated Stream - Sept 27-28, 2000

Instances inherit properties from other instances - part II

> dynamic property values and types
> dynamic property add & remove
> dynamic "change propagation" for everything that's dynamic,
> dynamic inheritance hierarchy
> selective value & type inheritance (container decides at run-time)
> single inheritance

This a different approach than last lecture, and **adds** capabilities to **dynamically add** (and remove) **properties that were not initially declared when the object class was defined.**.

It uses a HashMap to associate the current properties and values with the object which is receiving the inherited properties.

"**Container-decides**" selective inheritance means that an author of a class can say "this property is not inheritable," i.e.,a container can be selective about what properties can be inherited from it.

"**Containee-decides**," means that an object can say "I want to inherit just color from object instance b0, and just background-color from b1," i.e., a containee can be selective about what properties it inherits.

```
// ...much elided; visibility modifiers have been dropped.

import java.awt.Color;

import java.util.Map;      // ...maps keys to values  [Map is an inferface]
import java.util.HashMap;  // ...implements Map with a hash table

class Length {
  int  pts;  // ...for simplicity, assume that pts are the universal units

  Length(int pts)  { this.pts = pts; }

  public String toString()
    { return String.valueOf(pts) + " pts"; }
}

class Element {
  // the local table of (propertyName, value) pairs...
  Map  localCSSPropertyValues         = new HashMap();

  // the local table of (propertyName, isInheritable) pairs...
  Map  localCSSPropertyIsInheritables = new HashMap();

  // the static table of default values...
  static Map defaultCSSPropertyValues = new HashMap();

  static {
    setCSSPropertyDefaultValue("color", Color.black);
```

```
    setCSSPropertyDefaultValue("padding-left", new Length(0));
  }

  // the static table of default isInheritable settings...
  static Map defaultCSSPropertyIsInheritables = new HashMap();

  static {
    setCSSPropertyDefaultIsInheritable("color", true);
    setCSSPropertyDefaultIsInheritable("padding-left", false);
  }

  Element container;  // ...a root element has container == null

  Element()
    { container = null; }

  Element getContainer()
    { return container; }

  void setContainer(Element container)
    { this.container = container; }

  boolean isRoot()
    { return (container == null); }

  Object getCSSProperty(String propertyName)
    { return getCSSProperty(propertyName, false); }

  Object getCSSProperty(String propertyName, boolean attemptingToInherit)
{
    Object value = localLookup(propertyName);

    if (value != null) {
      if (attemptingToInherit && ! isInheritable(propertyName))
        // not inheritable...
        return getCSSPropertyDefaultValue(propertyName);

      // override container...
      return value;
        // [...more precisely, local value takes precedence over any values
        //  that might be defined further up in the container hierarchy.]
    }

    if (container == null)
      // no value defined...
      return getCSSPropertyDefaultValue(propertyName);

    // attempt to inherit value from container...
    return container.getCSSProperty(propertyName, true);
  }

  void setCSSProperty(String propertyName, Object value) {
    if (value == null)
      { unsetCSSProperty(propertyName);  return; }
```

```
    localCSSPropertyValues.put(propertyName, value);

    // note that the isInheritable flag is not modified in
    //  this case...
  }

  void setCSSProperty(String propertyName, Object value,
                      boolean isInheritable) {
    setCSSProperty(propertyName, value);

    if (value != null)
      localCSSPropertyIsInheritables.put(propertyName,
        new Boolean(isInheritable));
  }

  void unsetCSSProperty(String propertyName) {
    localCSSPropertyValues        .remove(propertyName);
    localCSSPropertyIsInheritables.remove(propertyName);
  }

  // methods that interact with the element's local tables...
  // [in a "real" version these would have restricted visibility.]

  Object localLookup(String propertyName)
    { return localCSSPropertyValues.get(propertyName); }

  boolean isInheritable(String propertyName) {
    Boolean b = (Boolean) localCSSPropertyIsInheritables.get(propertyName);

    if (b != null)  { return b.booleanValue(); }

    return getCSSPropertyDefaultIsInheritable(propertyName);
  }

  // methods for maintaining the static defaults...

  static Object getCSSPropertyDefaultValue(String propertyName)
    { return defaultCSSPropertyValues.get(propertyName); }

  static void setCSSPropertyDefaultValue(String propertyName,
           Object value)
    { defaultCSSPropertyValues.put(propertyName, value); }

  static boolean getCSSPropertyDefaultIsInheritable(String propertyName) {
    Boolean b = (Boolean)
defaultCSSPropertyIsInheritables.get(propertyName);
    return (b != null) ? b.booleanValue() : true;

    // ...so, properties are inheritable, unless specified otherwise.
  }

  static void setCSSPropertyDefaultIsInheritable(String propertyName,
                                                 boolean isInheritable) {
```

```
    defaultCSSPropertyIsInheritables.put(propertyName,
      new Boolean(isInheritable));
  }

  // ...a "real" version would most likely need a property iterator...
}

class BlockElement  extends Element { /* ... */ }
class InlineElement extends Element { /* ... */ }

class HTMLElement   extends Element { /* ... */ }
class BodyElement   extends Element { /* ... */ }

class ParagraphElement     extends  BlockElement { /* ... */ }
class UnorderedListElement extends  BlockElement { /* ... */ }
class ListItemElement      extends  BlockElement { /* ... */ }
class BoldElement          extends InlineElement { /* ... */ }

class ElementTest {
  public static void main(String[] as) {
    // <html>                                              <!-- html -->
    //   <body>                                            <!-- body -->
    //     <p style="color: red; padding-left: 10pt">      <!--  p0  -->
    //       <b>bold and red</b>                           <!--  b0  -->
    //       <ul style="color: green">                     <!-- ul0  -->
    //         <li> <b>bold and green</b>                  <!-- li0 & b1 -->
    //       </ul>
    //     </p>
    //   </body>
    // </html>

    HTMLElement           html = new HTMLElement();
    BodyElement           body = new BodyElement();
    ParagraphElement      p0   = new ParagraphElement();
    BoldElement           b0   = new BoldElement();
    UnorderedListElement  ul0  = new UnorderedListElement();
    ListItemElement       li0  = new ListItemElement();
    BoldElement           b1   = new BoldElement();

    body.setContainer(html);
    p0  .setContainer(body);
    b0  .setContainer(p0);
    ul0 .setContainer(p0);
    li0 .setContainer(ul0);
    b1  .setContainer(li0);

    p0  .setCSSProperty("color", Color.red);
    p0  .setCSSProperty("padding-left", new Length(10));

    ul0 .setCSSProperty("color", Color.green);

    System.out.print ("b0.color has-value ");
    System.out.println(b0.getCSSProperty("color"));  // red
                                                     //   (inherited from
p0)
```

```
    System.out.print ("b1.color has-value ");
    System.out.println(b1.getCSSProperty("color"));  // green
                                                     //   (inherited from
ul0)

    System.out.print ("p0 .padding-left has-value ");
    System.out.println(p0 .getCSSProperty("padding-left"));  // 10 pts (local)
    System.out.print ("ul0.padding-left has-value ");
    System.out.println(ul0.getCSSProperty("padding-left"));  //  0 pts
                                                             //  (not inherited)
```

// properties can be dynamically modified...

```
    System.out.println();

    System.out.println("p0.color = blue");
    p0 .setCSSProperty("color", Color.blue);

    System.out.print ("b0.color has-value ");
    System.out.println(b0.getCSSProperty("color"));  // blue
                                                     //(inherited from p0)

      // ...i.e., "bold and red" is now in blue.
```

// ...and properties can likewise be dynamically overridden...

```
    System.out.println("b0.color = red");
    b0 .setCSSProperty("color", Color.red);

    System.out.print ("b0.color has-value ");
    System.out.println(b0.getCSSProperty("color"));  // red
                                          //  (local value overrides p0)

      // ...i.e., "bold and red" is back to red.

    System.out.println("unset b0.color");
    b0 .setCSSProperty("color", null);

    System.out.print ("b0.color has-value ");
    System.out.println(b0.getCSSProperty("color"));  // blue
                                                  //   (inherited from p0)

      // ...i.e., "bold and red" is blue again.

    System.out.println("p0.color = red");
    p0 .setCSSProperty("color", Color.red);

    System.out.print ("b0.color has-value ");
    System.out.println(b0.getCSSProperty("color"));  // red
                                                  //   (inherited from p0)

      // ...i.e., "bold and red" is back to red.
```

// it's possible to add and remove properties dynamically...

```
System.out.println();

Element.setCSSPropertyDefaultValue("behavior", "no-op");

ul0.setCSSProperty("behavior", "javascript:closeWindow()");

System.out.print ("b1.behavior has-value ");
System.out.println(b1.getCSSProperty("behavior"));
                                        // closeWindow() (inherited from ul0)

ul0.unsetCSSProperty("behavior");

System.out.print ("b1.behavior has-value ");
System.out.println(b1.getCSSProperty("behavior"));  // no-op
                                                    // (default value)

// ...and a property's "type" can change at run-time...

System.out.println();

ul0.setCSSProperty("behavior", "javascript:closeWindow()");

System.out.print ("b1.behavior has-value ");
System.out.println(b1.getCSSProperty("behavior"));
                             // javascript:closeWindow()  (inherited from ul0)

  // ...i.e., b1.behavior is a String.

Runnable hello = new Runnable() {
    public void run() { System.out.println("hello"); }
    public String toString()
      { return "<a Runnable that prints \"hello\">"; }
  };

ul0.setCSSProperty("behavior", hello);

System.out.print ("b1.behavior has-value ");
System.out.println(b1.getCSSProperty("behavior"));
                  // <a Runnable that prints "hello"  // (inherited from ul0)

  // ...i.e., b1.behavior is an anonymous Runnable.

ul0.unsetCSSProperty("behavior");
```

// the inheritance hierarchy itself is also dynamic...

```
System.out.println();

System.out.println("b0.container = li0");
b0 .setContainer(li0);
System.out.println("b1.container = p0");
b1 .setContainer(p0);
```

```
      System.out.print ("b0.color has-value ");
      System.out.println(b0.getCSSProperty("color"));  // green
                                                       //  (inherited from ul0)
      System.out.print ("b1.color has-value ");
      System.out.println(b1.getCSSProperty("color"));  // red
                              //(inherited from  p0)

    // ...i.e., "bold and green" is now in red, and "bold and red"
    //  is in green, :).
  }
}
// actually, System.out.println(b0.getCSSProperty("color")) prints...
// java.awt.Color[r=255,g=0,b=0]
```