COMP 303 - Lecture Notes for Week 7 - Java Object Model

- Slides edited from, Object-Oriented Design Patterns, by Cay S. Horstmann
- Original slides available from: http://www.horstmann.com/design_and_patterns.html
- Modifications made by Laurie Hendren, McGill University
- Topics this week:
  - The Java Type System
  - Type Inquiry
  - The Object Class
  - Shallow and Deep Copy
  - Serialization
  - Reflection
  - The Java Beans Component Model

next .... [Slide 1] ....
Types

- Type: set of values and the operations that can be applied to the values
- Strongly typed language: compiler and run-time system check that no operation can execute that violates type system rules
- Compile-time check
  ```java
  Employee e = new Employee();
e.clear();  // ERROR
  ```
- Run-time check:
  ```java
  e = null;
e.setSalary(20000);  // ERROR
  ```
Java Types and Values

Types

- Primitive types:
  - int
  - short
  - long
  - byte
  - char
  - float
  - double
  - boolean
- Class types
- Interface types
- Array types
- The null type
- Note: void is not a type

Java Values

- value of primitive type
- reference to object of class type
- reference to array
- null
- Note: Can’t have value of interface type
Subtype Relationship

S is a subtype of T if

- S and T are the same type
- S and T are both class types, and T is a direct or indirect superclass of S
- S is a class type, T is an interface type, and S or one of its superclasses implements T
- S and T are both interface types, and T is a direct or indirect superinterface of S
- S and T are both array types, and the component type of S is a subtype of the component type of T
- S is not a primitive type and T is the type Object
- S is an array type and T is Cloneable or Serializable
- S is the null type and T is not a primitive type
The ArrayStoreException

- Rectangle[] is a subtype of Shape[]
- Can assign Rectangle[] value to Shape[] variable:

```java
Rectangle[] r = new Rectangle[10];
Shape[] s = r;
```

- Both r and s are references to the same array
- That array holds rectangles
- The assignment
  ```java
  s[0] = new Polygon();
  ```
  compiles
- Throws an ArrayStoreException at runtime
- Each array remembers its component type
**Wrapper Classes**

- Primitive types aren’t classes
- Use wrappers when objects are expected
- Wrapper for each type:

  Integer  Short  Long  Byte
  Character  Float  Double  Boolean

- **Example: ArrayList**

  ```java
  ArrayList numbers = new ArrayList();
  numbers.add(new Integer(13));
  int n = ((Integer)numbers.get(0)).intValue();
  ```
Enumerated Types

- Finite set of values
- Example: \{ SMALL, MEDIUM, LARGE \}
- Java has no syntax for enumerated types
- Can fake them with integer constants
  
  ```java
  public static final int SMALL = 1;
  public static final int MEDIUM = 2;
  public static final int LARGE = 3;
  ```
- Not typesafe
  
  ```java
  int size = LARGE;
  size++;  
  ```
**Typesafe Enumerations**

- Class with fixed number of instances

```java
public class Size {
    private Size(String name) {
        this.name = name;
    }
    private String name;
    public static final Size SMALL = new Size("SMALL");
    public static final Size MEDIUM = new Size("MEDIUM");
    public static final Size LARGE = new Size("LARGE");
}
```

- Private constructor!
- String field not necessary (but convenient for `toString`, serialization)
- Typical use:

```java
Size imageSize = Size.MEDIUM;
if (imageSize == Size.SMALL) . . .
```

---

previous | start | next .... [Slide 8] ....
Type Inquiry

- Test whether e is a Shape:
  
  ```java
  if (e instanceof Shape) . . .
  ```
- Common before casts:
  
  ```java
  Shape s = (Shape) e;
  ```
- Don’t know exact type of e
- Could be any class implementing Shape
- If e is null, test returns false (no exception)
The **Class Class**

- **getClass** method gets class of any object
- Returns object of type **Class**:
- **Class** object describes a **type**

```java
Object e = new Rectangle();
Class c = e.getClass();
System.out.println(c.getName()); // prints java.awt.Rectangle
```

- **Class.forName** method yields **Class** object:

```java
Class c = Class.forName(java.awt.Rectangle);
```

- **.class** suffix yields **Class** object:

```java
Class c = Rectangle.class; // java.awt prefix not needed
```

- **Class** is a misnomer: int.class, void.class, Shape.class
An Employee Object vs. the Employee.class Object

: Employee
   name = "Jane Doe"
   salary = 50000

: Class
   name = "Employee"
   superclass = 

: Class
   name = "java.lang.Object"
   superclass = null
Type Inquiry

- Test whether \( e \) is a Rectangle:
  
  ```java
  if (e.getClass() == Rectangle.class) . . .
  ```

- Ok to use `==`
- A unique `Class` object for every class
- Test fails for subclasses
- Use `instanceof` to test for subtypes:
  ```java
  if (x instanceof Rectangle) . . .
  ```
Array Types

- Can apply `getClass` to an array
- Returned object describes an array type

```java
double[] a = new double[10];
Class c = a.getClass();
if (c.isArray())
    System.out.println(c.getComponentType());
    // prints double
```

- `getName` produces strange names for array types

```
[D for double[]]
[[java.lang.String; for String[][]
```
Object: The Cosmic Superclass

- All classes extend Object
- Most useful methods:
  - String toString()
  - boolean equals(Object otherObject)
  - Object clone()
  - int hashCode()
The toString Method

- Returns a string representation of the object
- Useful for debugging
- Example: Rectangle.toString returns something like
  `java.awt.Rectangle[x=5,y=10,width=20,height=30]`
- toString used by concatenation operator
- `aString + anObject` means
  `aString + anObject.toString()`
- `Object.toString` prints class name and object address
  `System.out.println(System.out)` yields
  `java.io.PrintStream@d2460bf`
- Implementor of PrintStream didn’t override toString:
Overriding the `toString` Method

- Format all fields:

  ```java
  public class Employee {
    public String toString() {
      return getClass().getName() + "[name= " + name
      + ", salary= " + salary
      + "]";
    }
  }
  ...
  }
  ```

- Typical string:

  Employee[name=Harry Hacker, salary=35000]
Overriding `toString` in Subclass

- Format superclass first

  ```java
  public class Manager extends Employee {
    public String toString() {
      return super.toString() + "[department=" + department + "]";
    }
    ...
  }
  ```

- Typical string

  ```text
  Manager[name=Dolly Dollar,salary=100000][department=Finance]
  ```

- Note that superclass reports actual class name
The **equals** Method

- **equals** tests for equal *contents*
- `==` tests for equal *location*
- Used in many standard library methods
- Example: `ArrayList.indexOf`

```java
/**
 * Searches for the first occurrence of the given argument, testing for equality using the equals method.
 * @param elem an object.
 * @return the index of the first occurrence of the argument in this list; returns -1 if the object is not found.
 */
public int indexOf(Object elem)
{
    if (elem == null)
    {
        for (int i = 0; i < size; i++)
            if (elementData[i] == null) return i;
    }
    else
    {
        for (int i = 0; i < size; i++)
            if (elem.equals(elementData[i])) return i;
    }
    return -1;
}
```
Overriding the **equals** Method

- Notion of equality depends on class
- Common definition: compare all fields

```java
public class Employee {
    public boolean equals(Object otherObject) {
        // not complete--see below
        Employee other = (Employee)otherObject;
        return name.equals(other.name) && salary == other.salary;
    }
    ...
}
```

- Must cast the **Object** parameter to subclass
- Use `==` for primitive types, `equals` for object fields
Overriding \texttt{equals} in Subclass

- Call \texttt{equals} on superclass

```java
public class Manager {
    public boolean equals(Object otherObject) {
        Manager other = (Manager)otherObject;
        return super.equals(other) && department.equals(other.department);
    }
}
```
Not all \texttt{equals} Methods are Simple

- Two sets are equal if they have the same elements \textit{in some order}

```java
public boolean equals(Object o) {
    if (o == this) return true;
    if (!(o instanceof Set)) return false;
    Collection c = (Collection) o;
    if (c.size() != size()) return false;
    return containsAll(c);
}
```
The **Object.equals** Method

- **Object.equals** tests for identity:

  ```java
  public class Object {
      public boolean equals(Object obj) {
          return this == obj;
      }
  }
  ```

- Override **equals** if you don’t want to inherit that behavior
Requirements for \texttt{equals} Method

- \textit{reflexive}: \texttt{x.equals(x)}
- \textit{symmetric}: \texttt{x.equals(y)} if and only if \texttt{y.equals(x)}
- \textit{transitive}: if \texttt{x.equals(y)} and \texttt{y.equals(z)}, then \texttt{x.equals(z)}
- \texttt{x.equals(null)} must return \texttt{false}
Fixing Employee.equals

- Violates two rules
- Add test for null:
  \[
  \text{if (otherObject == null) return false}
  \]
- What happens if otherObject not an Employee
- Should return false (because of symmetry)
- Common error: use of instanceof
  \[
  \text{if (!(otherObject instanceof Employee)) return false;}
  \]
  // don’t do this for non-final classes
- Violates symmetry: Suppose e, m have same name, salary
  e.equals(m) is true (because m instanceof Employee)
  m.equals(e) is false (because e isn’t an instance of Manager)
- Remedy: Test for class equality
  \[
  \text{if (getClass() != otherObject.getClass()) return false;}
  \]
The Perfect `equals` Method

- Start with these three tests:

  ```java
  public boolean equals(Object otherObject)
  {
      if (this == otherObject) return true;
      if (otherObject == null) return false;
      if (getClass() != otherObject.getClass()) return false;
      ...
  }
  ```

- First test is an optimization
Hashing

- \texttt{hashCode} method used in HashMap, HashSet
- Computes an \texttt{int} from an object
- Example: hash code of String
  
  \begin{verbatim}
  int h = 0;
  for (int i = 0; i < s.length(); i++)
      h = 31 * h + s.charAt(i);
  \end{verbatim}
  
  - Hash code of "eat" is 100184
  - Hash code of "tea" is 114704
Hashing

- Must be compatible with equals:
  \[ \text{if } x \text{.equals}(y) \text{, then } x \text{.hashCode()} == y \text{.hashCode()} \]
- `Object.hashCode` hashes memory address
- *NOT* compatible with redefined `equals`
- Remedy: Hash all fields and combine codes:

```java
public class Employee {
    public int hashCode()
    {
        return name.hashCode()
            + new Double(salary).hashCode();
    }
    ...
}
```

```
Shallow and Deep Copy

- Assignment (copy = e) makes shallow copy
- Clone to make deep copy
- Employee cloned = (Employee)e.clone();
Cloning

e =

: Employee

name = "Smith"
salary = 35000

cloned =

: Employee

name = "Smith"
salary = 35000
Cloning

- `Object.clone` makes new object and copies all fields
- Cloning is subtle
- `Object.clone` is protected
- Subclass must redefine `clone` to be `public`

```java
public class Employee {
    public Object clone() {
        return super.clone(); // not complete
    }
    ...
}
```
The Cloneable Interface

- Object.clone is nervous about cloning
- Will only clone objects that implement Cloneable interface

    public interface Cloneable
    {
    }

- Interface has no methods!
- Tagging interface--used in test
    if x implements Cloneable
- Object.clone throws CloneNotSupportedException
- A checked exception
The clone Method

```java
public class Employee
    implements Cloneable
{
    public Object clone()
    {
        try
        {
            return super.clone();
        }
        catch(CloneNotSupportedException e)
        {
            return null; // won’t happen
        }
    }
    ...
}
```
Shallow Cloning

- `clone` makes a shallow copy
- Instance fields aren’t cloned

Previous | Start | Next .... [Slide 33] ....
Deep Cloning

- Why doesn’t `clone` make a deep copy?
  - Wouldn’t work for cyclic data structures
- Not a problem for immutable fields
- You must clone mutable fields

```java
public class Employee
    implements Cloneable
{
    public Object clone()
    {
        try
        {
            Employee cloned = (Employee)super.clone();
            cloned.hireDate = (Date)hiredate.clone();
            return cloned;
        }
        catch(CloneNotSupportedException e)
        {
            return null; // won’t happen
        }
    }
    ...
}
```
Deep Cloning

```
<table>
<thead>
<tr>
<th>e =</th>
</tr>
</thead>
<tbody>
<tr>
<td>: Employee</td>
</tr>
<tr>
<td>name =</td>
</tr>
<tr>
<td>salary = 35000</td>
</tr>
<tr>
<td>hireDate =</td>
</tr>
<tr>
<td>: String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cloned =</th>
</tr>
</thead>
<tbody>
<tr>
<td>: Employee</td>
</tr>
<tr>
<td>name =</td>
</tr>
<tr>
<td>salary = 35000</td>
</tr>
<tr>
<td>hireDate =</td>
</tr>
<tr>
<td>: Date</td>
</tr>
<tr>
<td>: Date</td>
</tr>
</tbody>
</table>
```

previous | start | next .... [Slide 35] ....
Cloning and Inheritance

- `Object.clone` is paranoid
  - clone is protected
  - clone only clones `Cloneable` objects
  - clone throws checked exception
- You don’t have that luxury
- `Manager.clone` must be defined if `Manager` adds mutable fields
- Rule of thumb: if you extend a class that defines `clone`, redefine `clone`
- Lesson to learn: Tagging interfaces are inherited. Use them only to tag properties that inherit
**Serialization**

- Save collection of objects to stream
  ```java
  Employee[] staff = new Employee[2];
  staff.add(new Employee(...));
  staff.add(new Employee(...));
  ```

- **Construct** `ObjectOutputStream`:
  ```java
  ObjectOutputStream out
  = new ObjectOutputStream(
      new FileOutputStream("staff.dat"));
  ```

- Save the array and close the stream
  ```java
  out.writeObject(staff);
  out.close();
  ```
Serialization

- The array *and all of its objects and their dependent objects* are saved
- Employee doesn’t have to define any method
- Needs to implement the `Serializable` interface
- Another tagging interface with no methods
How Serialization Works

- Each newly encountered object is saved
- Each object gets a serial number in the stream
- No object is saved twice
- Reference to already encountered object saved as "reference to #"
Serializing Unserializable Classes

- Some classes are not serializable
- Example: `Ellipse2D.Double`
- How can we serialize `Car`?
- Suppress default serialization to avoid exception
- Mark with `transient`:
  ```java
  private transient Ellipse2D frontTire;
  ```
- Supply private (!) methods
  ```java
  private void writeObject(ObjectOutputStream out) {
  private void readObject(ObjectInputStream in) {
  ```
- In these methods
  - Call `writeDefaultObject/readDefaultObject`
  - Manually save other data
- `Ch7/serial/Car.java`
```java
import java.awt.*;
import java.awt.geom.*;
import java.io.*;

/**
 * A serializable car shape.
 */
public class Car implements Serializable {
    /**
     * Constructs a car.
     * @param x the left of the bounding rectangle
     * @param y the top of the bounding rectangle
     * @param width the width of the bounding rectangle
     */
    public Car(int x, int y, int width) {
        body = new Rectangle(x, y + width / 6,
                              width - 1, width / 6);
        roof = new Rectangle(x + width / 3, y,
                             width / 3, width / 6);
        frontTire = new Ellipse2D.Double(x + width / 6, y + width / 3,
                                          width / 6, width / 6);
        rearTire = new Ellipse2D.Double(x + width * 2 / 3, y + width / 3,
                                         width / 6, width / 6);
    }

    private void writeObject(ObjectOutputStream out)
            throws IOException {
        out.defaultWriteObject();
        writeRectangularShape(out, frontTire);
        writeRectangularShape(out, rearTire);
    }

    /*
     * A helper method to write a rectangular shape.
     * @param out the stream onto which to write the shape
     * @param s the shape to write
     */
    private static void writeRectangularShape(ObjectOutputStream out,
                                               Object s)
```
RectangularShape s)
   throws IOException
{
   out.writeDouble(s.getX());
   out.writeDouble(s.getY());
   out.writeDouble(s.getWidth());
   out.writeDouble(s.getHeight());
}

private void readObject(ObjectInputStream in)
   throws IOException, ClassNotFoundException
{
   in.defaultReadObject();
   frontTire = new Ellipse2D.Double();
   readRectangularShape(in, frontTire);
   rearTire = new Ellipse2D.Double();
   readRectangularShape(in, rearTire);
}

/**
 * A helper method to read a rectangular shape.
 * @param in the stream from which to read the shape
 * @param s the shape to read. The method sets the frame
 * of this rectangular shape.
 */
private static void readRectangularShape(ObjectInputStream in,
   RectangularShape s)
   throws IOException
{
   double x = in.readDouble();
   double y = in.readDouble();
   double width = in.readDouble();
   double height = in.readDouble();
   s.setFrame(x, y, width, height);
}

/**
 * Draws the car.
 * @param g2 the graphics context
 */
public void draw(Graphics2D g2)
private static String formatRectangularShape(RectangularShape s) {
    return s.getClass().getName()
            + "[x=" + s.getX() + ",y=" + s.getY() + ",width=" + s.getWidth() + ",height=" + s.getHeight() + "]";
}

private Rectangle body;
private Rectangle roof;
private transient Ellipse2D.Double frontTire;
private transient Ellipse2D.Double rearTire;

Reflection

- Ability of running program to find out about its objects and classes
- **Class** object reveals
  - superclass
  - interfaces
  - package
  - names and types of fields
  - names, parameter types, return types of methods
  - parameter types of constructors
Reflection

- Class getSuperclass()
- Class[] getInterfaces()
- Package getPackage()
- Field[] getDeclaredFields()
- Constructor[] getDeclaredConstructors()
- Method[] getDeclaredMethods()

Example: Enumerating static fields Math class

Field[] fields = Math.class.getDeclaredFields();
for (int i = 0; i < fields.length; i++)
    if (Modifier.isStatic(fields[i].getModifiers()))
        System.out.println(fields[i].getName());
Enumerating Constructors

- Print the names and parameter types of all Rectangle constructors:

  ```java
  for (int i = 0; i < cons.length; i++)
  {
      Class[] params = cons[i].getParameterTypes();
      System.out.print("Rectangle(");
      for (int j = 0; j < params.length; j++)
      {
          if (j > 0) System.out.print(",
          System.out.print(params[j].getName());
      }
      System.out.println(
  }
  }
  ```

- Yields

  Rectangle()
  Rectangle(java.awt.Rectangle)
  Rectangle(int, int, int, int)
  Rectangle(int, int)
  Rectangle(java.awt.Point, java.awt.Dimension)
  Rectangle(java.awt.Point)
  Rectangle(java.awt.Dimension)
Getting A Single Method Descriptor

- Supply method name
- Supply array of parameter types
- Example: Get Rectangle.contains(int, int):

```java
Method m = Rectangle.class.getDeclaredMethod(
    "contains",
    new Class[] { int.class, int.class });
```

- Example: Get default Rectangle constructor:

```java
Constructor c = Rectangle.class.getDeclaredConstructor(
    new Class[] {});
```

Invoking a Method

- Supply implicit parameter (null for static methods)
- Supply array of explicit parameter values
- Wrap primitive types
- Unwrap primitive return value
- Example: Call System.out.println("Hello, World") the hard way.

```
Method m = PrintStream.class.getDeclaredMethod("println",
    new Class[] { String.class });
    m.invoke(System.out,
    new Object[] { "Hello, World!" });
```
Inspecting Objects

- Can obtain object contents at runtime
- Useful for generic debugging tools
- Need to gain access to private fields

```java
Class c = obj.getClass();
Field f = c.getDeclaredField(name);
f.setAccessible(true);
```

- Throws exception if security manager disallows access
- Access field value:

```java
Object value = f.get(obj);
f.set(obj, value);
```

- Use wrappers for primitive types
Inspecting Objects

- Example: Peek inside string tokenizer
- Ch7/code/reflect2/FieldTest.java
- Output

```java
int currentPosition=0
int newPosition=-1
int maxPosition=13
java.lang.String str="Hello, World!"
java.lang.String delimiters=,
boolean retDelims=false
boolean delimsChanged=false
char maxDelimChar=,
---
int currentPosition=5
...
```
import java.lang.reflect.*;
import java.util.*;

/**
 * This program shows how to use reflection to print the names and values of all fields of an object.
 */
public class FieldTest {
    public static void main(String[] args) throws IllegalAccessException {
        String input = "Hello, World!";
        StringTokenizer tokenizer = new StringTokenizer(input, ",");
        System.out.println(spyFields(tokenizer));
        tokenizer.nextToken();
        System.out.println("\nAfter calling nextToken:\n");
        System.out.println(spyFields(tokenizer));
    }

    /**
     * Spies on the field names and values of an object.
     * @param obj the object whose fields to format
     * @return a string containing the names and values of all fields of obj
     */
    public static String spyFields(Object obj) throws IllegalAccessException {
        StringBuffer buffer = new StringBuffer();
        Field[] fields = obj.getClass().getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            Field f = fields[i];
            f.setAccessible(true);
            Object value = f.get(obj);
            buffer.append(f.getType().getName());
            buffer.append(" ");
            buffer.append(f.getName());
            buffer.append("=");
            buffer.append("" + value);
        }
        return buffer.toString();
    }
}
42:          buffer.append("\n");
43:      }
44:          return buffer.toString();
45:      }
46:   }
Inspecting Array Elements

- Use static methods of `Array` class

  ```java
  Object value = Array.get(a, i);
  Array.set(a, i, value);
  
  int n = Array.getLength(a);
  ```

- Construct new array:

  ```java
  Object a = Array.newInstance(type, length);
  ```
Components

- More functionality than a single class
- Reuse and customize in multiple contexts
- "Plug components together" to form applications
- Successful model: Visual Basic controls
- Examples:
  - calendar
  - graph
  - database
  - link to robot or instrument
- Components composed into program inside builder environment
A Builder Environment
Java Beans

- Java component model
- Bean has:
  - methods (just like classes)
  - properties
  - events
A Calendar Bean

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

previous | start | next .... [Slide 52] ....
A Property Sheet

- Edit properties with *property sheet*

![Property Sheet Diagram]

previous | start | next .... [Slide 53] ....
Facade Class

- Bean usually composed of multiple classes
- One class nominated as *facade class*
- Clients use only facade class methods

Facade Pattern

Context

1. A subsystem consists of multiple classes, making it complicated for clients to use
2. Implementor may want to change subsystem classes
3. Want to give a coherent entry point

Solution

1. Define a facade class that exposes all capabilities of the subsystem as methods
2. The facade methods delegate requests to the subsystem classes
3. The subsystem classes do not know about the facade class
Facade Pattern

Client \rightarrow Facade

Subsystem Class

Subsystem Class

Subsystem Class

Subsystem Class
## Facade Pattern

<table>
<thead>
<tr>
<th>Name in Design Pattern</th>
<th>Actual Name (Beans)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Builder tool</td>
</tr>
<tr>
<td>Facade</td>
<td>Main bean class with which the tool interacts</td>
</tr>
<tr>
<td>SubsystemClass</td>
<td>Class used to implement bean functionality</td>
</tr>
</tbody>
</table>
Bean Properties

- Property = value that you can get and/or set
- Most properties are get-and-set
- Can also have get-only and set-only
- Property not the same as instance field
- Setter can set fields, then call repaint
- Getter can query database

Property Syntax

- Not Java :-(
- C#, JavaScript, Visual Basic
- `b.propertyName = value`
calls setter
- `variable = b.propertyName`
calls getter
Java Naming Conventions

- property = pair of methods
  
  public X getPropertyName()
  
  public void setPropertyName(X newValue)

- Replace propertyName with actual name
  (e.g. getColor/setColor)

- Exception for boolean properties:
  
  public boolean isPropertyName()

- Decapitalization hokus-pokus:
  
  getColor -> color
  
  getURL -> URL
Editing Beans in a Builder Tool

- Use wizard to make empty frame
Editing Beans in a Builder Tool

- Add button to frame, then edit button with property sheet.
Packaging a Bean

- Compile bean classes
  Ch7/carbean1/CarBean.java
- Create manifest file
  Ch7/carbean1/CarBean.mf
- Run JAR tool:
  `jar cvfm CarBean.jar CarBean.mf *.class`
- Import JAR file into builder environment
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

/**
 * A panel that draws a car shape.
 */
public class CarBean extends JPanel {

    /**
     * Constructs a default car bean.
     */
    public CarBean() {
        x = 0;
        y = 0;
        width = DEFAULT_CAR_WIDTH;
        height = DEFAULT_CAR_HEIGHT;
    }

    /**
     * Sets the x property.
     * @param newValue the new x position
     */
    public void setX(int newValue) {
        x = newValue;
        repaint();
    }

    /**
     * Gets the x property.
     * @return the x position
     */
    public int getX() {
        return x;
    }

    /**
     * Sets the y property.
     */
    public void setY(int newValue) {
        y = newValue;
        repaint();
    }

    /**
     * Gets the y property.
     * @return the y position
     */
    public int getY() {
        return y;
    }
}
@param newValue the new y position

public void setY(int newValue)
{
    y = newValue;
    repaint();
}

/**
 * Gets the y property.
 * @return the y position
 */
public int getY()
{
    return y;
}

public void paintComponent(Graphics g)
{
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    Rectangle2D.Double body
        = new Rectangle2D.Double(x, y + height / 3, width - 1, height / 3);
    Ellipse2D.Double frontTire
        = new Ellipse2D.Double(x + width / 6, y + height * 2 / 3, height / 3, height / 3);
    Ellipse2D.Double rearTire
        = new Ellipse2D.Double(x + width * 2 / 3, y + height * 2 / 3, height / 3, height / 3);

    // the bottom of the front windshield
    Point2D.Double r1
        = new Point2D.Double(x + width / 6, y + height / 3);
    // the front of the roof
    Point2D.Double r2
        = new Point2D.Double(x + width / 3, y);
    // the rear of the roof
    Point2D.Double r3
        = new Point2D.Double(x + width * 2 / 3, y);
    // the bottom of the rear windshield
083:       Point2D.Double r4
084:           = new Point2D.Double(x + width * 5 / 6, y + height / 3);
085: 
086:       Line2D.Double frontWindshield
087:           = new Line2D.Double(r1, r2);
088:       Line2D.Double roofTop
089:           = new Line2D.Double(r2, r3);
090:       Line2D.Double rearWindshield
091:           = new Line2D.Double(r3, r4);
092: 
093:           g2.draw(body);
094:           g2.draw(frontTire);
095:           g2.draw(rearTire);
096:           g2.draw(frontWindshield);
097:           g2.draw(roofTop);
098:           g2.draw(rearWindshield);
099:       }
100: 
101:       public Dimension getPreferredSize()
102:       {
103:           return new Dimension(DEFAULT_PANEL_WIDTH,
104:                           DEFAULT_PANEL_HEIGHT);
105:       }
106: 
107:       private int x;
108:       private int y;
109:       private int width;
110:       private int height;
111: 
112:       private static final int DEFAULT_CAR_WIDTH = 60;
113:       private static final int DEFAULT_CAR_HEIGHT = 30;
114:       private static final int DEFAULT_PANEL_WIDTH = 160;
115:       private static final int DEFAULT_PANEL_HEIGHT = 130;
116:       }
Composing Beans

- Make new frame
- Add car bean, slider to frame
- Edit stateChanged event of slider
- Add handler code
  ```java
  carBean1.setX(jSlider1.getValue());
  ```
- Compile and run
- Move slider: the car moves
Bean Information

- Builder environment loads beans
- Looks for get/set methods in facade class
- Can discover spurious properties
  ```java
  JButton: Object getTreeLock()
  ```
- Alternate mechanism: BeanInfo class
- Must have name *FacadeClassName*BeanInfo
- E.g. HouseBeanBeanInfo
The BeanInfo Interface

Image getIcon(int iconKind)
BeanDescriptor getBeanDescriptor()
MethodDescriptor[] getMethodDescriptors()
PropertyDescriptor[] getPropertyDescriptors()
EventSetDescriptor[] getEventSetDescriptors()
int getDefaultEventIndex()
int getDefaultPropertyIndex()
BeanInfo[] getAdditionalBeanInfo()
Removing Spurious Properties

class MyBeanBeanInfo extends SimpleBeanInfo
{
    public PropertyDescriptor[] getPropertyDescriptors()
    {
        try
        {
            return new PropertyDescriptor[]
            {
                new PropertyDescriptor("x", CarBean.class);
                new PropertyDescriptor("y", CarBean.class);
            };
        }
        catch (IntrospectionException exception)
        {
            return null;
        }
    }
}

previous | start | next .... [Slide 65] ....
Property Editors

- Property sheet enumerates properties
- Allows user to edit property values
- How can one edit values of arbitrary types?
- Built-in editors for String, Color, etc
- Supply custom editor for your own types
Custom Property Editors

- Three kinds of editors
  - Text
  - Finite set of choices
  - Arbitrary painting and editing
- Implement PropertyEditor interface
- Or extend PropertyEditorSupport class

Editing Text Properties

- Convert between your type and String
- Define two methods

  ```java
  public String getAsText()
  public void setAsText(String s)
  ```

- Property sheet uses text field

Editing Choice Properties

- Your type has finite set of string choices
- E.g. DrawMode.DRAW, DrawMode.FILL
- String[] getTags() returns array of choices
- Also need to define getAsText/setAsText
- Property sheet uses combo box
Editing Arbitrary Properties

- Your type isn’t easily editable as string
- E.g. Color
- Property editor pops up your edit dialog
  - boolean supportsCustomEditor() must return true
  - Component getCustomEditor() returns dialog
- Property editor can paint current value of your type
  - boolean isPaintable() must return true
  - void paintValue(Graphics g, Rectangle bounds) paints

previous | start | next .... [Slide 68] ....
Registering Property Editors

- Global setting
  
  `PropertyEditorManager.registerEditor(valueClass, editorClass)`

- Per-bean setting
  In bean info class:

  ```java
  PropertyDescriptor dimensionProperty = new PropertyDescriptor("dimension", CarBean.class);
  dimensionProperty.setPropertyEditorClass(DimensionEditor.class);
  ```
Example: CarBean

- Ch7/carbean2/CarBean.java
- Ch7/carbean2/CarBeanBeanInfo.java
- Ch7/carbean2/DimensionEditor.java
- Ch7/carbean2/DrawMode.java
- Ch7/carbean2/DrawModeEditor.java
- Ch7/carbean2/CustomColorEditor.java
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

/**
 * A panel that draws a car shape.
 */
public class CarBean extends JPanel {

    public CarBean() {
        x = 0;
        y = 0;
        width = DEFAULT_WIDTH;
        height = DEFAULT_HEIGHT;
        color = DEFAULT_COLOR;
        fill = false;
    }

    public void setColor(Color color) {
        this.color = color;
        repaint();
    }

    public Color getColor() { return color; }

    /****************
    * Sets the color property.
    * @param color the new color
    */
    /****************
    * Gets the color property.
    * @return the current color
    */
    /****************
    * Sets the dimension property.
    * @param dimension the new dimension of the house
    */
public void setDimension(Dimension dimension) {
    width = (int) dimension.getWidth();
    height = (int) dimension.getHeight();
    repaint();
}

/**
   * Gets the dimension property.
   * @return the current dimension of the house
   */
public Dimension getDimension() {
    return new Dimension(width, height);
}

/**
   * Sets the drawMode property.
   * @param drawMode the new drawMode (DRAW or FILL)
   */
public void setDrawMode(DrawMode drawMode) {
    if (drawMode == DrawMode.DRAW) {
        fill = false;
        repaint();
    }
    else if (drawMode == DrawMode.FILL) {
        fill = true;
        repaint();
    }
}

/**
   * Gets the drawMode property.
   * @return the current drawMode (DRAW or FILL)
   */
public DrawMode getDrawMode() {

if (fill) return DrawMode.FILL;
else return DrawMode.DRAW;

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    Rectangle2D.Double body
        = new Rectangle2D.Double(x, y + height / 3,
                                  width - 1, height / 3);
    Ellipse2D.Double frontTire
        = new Ellipse2D.Double(x + width / 6,
                               y + height * 2 / 3, height / 3, height / 3);
    Ellipse2D.Double rearTire
        = new Ellipse2D.Double(x + width * 2 / 3,
                               y + height * 2 / 3, height / 3, height / 3);

    // the bottom of the front windshield
    Point2D.Double r1
        = new Point2D.Double(x + width / 6, y + height / 3);
    // the front of the roof
    Point2D.Double r2
        = new Point2D.Double(x + width / 3, y);
    // the rear of the roof
    Point2D.Double r3
        = new Point2D.Double(x + width * 2 / 3, y);
    // the bottom of the rear windshield
    Point2D.Double r4
        = new Point2D.Double(x + width * 5 / 6, y + height / 3);
    Line2D.Double frontWindshield
        = new Line2D.Double(r1, r2);
    Line2D.Double roofTop
        = new Line2D.Double(r2, r3);
    Line2D.Double rearWindshield
        = new Line2D.Double(r3, r4);

    g2.setColor(color);
    if (fill)
        g2.fill(body);
g2.fill(frontTire);
g2.fill(rearTire);
}
else {
g2.draw(body);
g2.draw(frontTire);
g2.draw(rearTire);
}
g2.draw(frontWindshield);
g2.draw(roofTop);
g2.draw(rearWindshield);
}

public static final int DRAW = 0;

public static final int FILL = 1;

private int x;

private int y;

private Color color;

private int width;

private int height;

private boolean fill;

private static final int DEFAULT_WIDTH = 50;

private static final int DEFAULT_HEIGHT = 80;

private static final Color DEFAULT_COLOR = Color.BLACK;
```java
import java.beans.*;

/**
The bean info for the car bean, specifying the properties and their editors.
*/
public class CarBeanBeanInfo extends SimpleBeanInfo {
    public PropertyDescriptor[] getPropertyDescriptors() {
        try {
            PropertyDescriptor colorProperty = new PropertyDescriptor("color",
                CarBean.class);
            colorProperty.setPropertyEditorClass(CustomColorEditor.class);

            PropertyDescriptor dimensionProperty = new PropertyDescriptor("dimension",
                CarBean.class);
            dimensionProperty.setPropertyEditorClass(DimensionEditor.class);

            PropertyDescriptor drawModeProperty = new PropertyDescriptor("drawMode",
                CarBean.class);
            drawModeProperty.setPropertyEditorClass(DrawModeEditor.class);

            return new PropertyDescriptor[] {
                colorProperty,
                dimensionProperty,
                drawModeProperty
            };
        }
        catch (IntrospectionException exception) {
        }
    }
```
return null;
```java
import java.awt.*;
import java.beans.*;
import java.util.*;

/**
 * A property editor for the Dimension type that presents
 * a dimension as a text "width x height".
 */
public class DimensionEditor extends PropertyEditorSupport {

    public String getAsText() {
        Dimension value = (Dimension) getValue();
        return (int) value.getWidth() + "x" + (int) value.getHeight();
    }

    public void setAsText(String s) {
        try {
            StringTokenizer tokenizer = new StringTokenizer(s, "x");
            if (!tokenizer.hasMoreTokens())
                throw new IllegalArgumentException();
            int width = Integer.parseInt(tokenizer.nextToken().trim());
            if (!tokenizer.hasMoreTokens())
                throw new IllegalArgumentException();
            int height = Integer.parseInt(tokenizer.nextToken().trim());
            setValue(new Dimension(width, height));
        } catch (NumberFormatException exception) {
            throw new IllegalArgumentException();
        }
    }
}
```
```java
import java.io.*;

public class DrawMode implements Serializable {
    private DrawMode(String name) { this.name = name; }
    private String name;
    protected Object readResolve() {
        if (name.equals("DRAW")) return DrawMode.DRAW;
        else if (name.equals("FILL")) return DrawMode.FILL;
        else return null;
    }
    public static final DrawMode DRAW = new DrawMode("DRAW");
    public static final DrawMode FILL = new DrawMode("FILL");
}
```
import java.beans.*;

/**
 * A property editor for the draw mode of the CarBean.
 */

public class DrawModeEditor extends PropertyEditorSupport {

    public String[] getTags() {
        return new String[] { "Draw", "Fill" };  
    }

    public String getAsText() {
        DrawMode value = (DrawMode) getValue();
        if (value == DrawMode.DRAW) return "Draw";
        if (value == DrawMode.FILL) return "Fill";
        return null;
    }

    public void setAsText(String s) {
        if (s.equals("Draw")) setValue(DrawMode.DRAW);
        else if (s.equals("Fill")) setValue(DrawMode.FILL);
        else throw new IllegalArgumentException();
    }
}
import java.awt.*;
import java.beans.*;
import javax.swing.*;
import javax.swing.event.*;

/**
 * A property editor for the Color type that uses a JColorChooser.
 */
public class CustomColorEditor extends PropertyEditorSupport {
    public String getAsText() {
        return null;
    }

    public boolean supportsCustomEditor() {
        return true;
    }

    public Component getCustomEditor() {
        final JColorChooser chooser = new JColorChooser();
        chooser.getSelectionModel().addChangeListener(new ChangeListener {
            public void stateChanged(ChangeEvent event) {
                setValue(chooser.getColor());
            }
        });
        return chooser;
    }

    public boolean isPaintable() {
        return true;
    }

    public void paintValue(Graphics g, Rectangle boundingBox) {
    }
}
Graphics2D g2 = (Graphics2D) g;
Color color = (Color) getValue();
g2.setColor(color);
g2.fill(boundingBox);
g2.setColor(Color.BLACK);
g2.draw(boundingBox);
Example: CarBean
Implementing a Property Sheet

- Used for graph framework in chapter 8
- Form shows property names on left, editors on right
Implementing a Property Sheet

- Get bean info for class
- Get properties from bean info
- Obtain property getter/setter methods from property descriptor
- Use these methods to read and write property values
- Each editor is text field, combo box, or button with painted icon
- Clicking on button brings up dialog
- Ch7/propedit/PropertySheet.java
- Ch7/propedit/PropertySheetTest.java
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.lang.reflect.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

/**
 * A component filled with editors for all editable properties
 * of an object.
 */
public class PropertySheet extends JPanel
{
    /**
     * Constructs a property sheet that shows the editable
     * properties of a given object.
     * @param object the object whose properties are being edited
     */
    public PropertySheet(Object bean)
    {
        try
        {
            BeanInfo info = Introspector.getBeanInfo(bean.getClass());
            PropertyDescriptor[] descriptors = info.getPropertyDescriptors();
            setLayout(new FormLayout());
            for (int i = 0; i < descriptors.length; i++)
            {
                PropertyEditor editor = getEditor(bean, descriptors[i]);
                if (editor != null)
                {
                    add(new JLabel(descriptors[i].getName()));
                    add(getEditorComponent(editor));
                }
            }
        }
        catch (IntrospectionException exception)
        {
        }
    }
}
public PropertyEditor getEditor(final Object bean, PropertyDescriptor descriptor) {
    try {
        Method getter = descriptor.getReadMethod();
        if (getter == null) return null;
        final Method setter = descriptor.getWriteMethod();
        if (setter == null) return null;
        final PropertyEditor editor;
        Class editorClass = descriptor.getPropertyEditorClass();
        if (editorClass != null)
            editor = (PropertyEditor) editorClass.newInstance();
        else
            editor = PropertyEditorManager.findEditor(
                descriptor.getPropertyType());
        if (editor == null) return null;
        Object value = getter.invoke(bean, new Object[] {});
        editor.setValue(value);
        editor.addPropertyChangeListener(new PropertyChangeListener() {
            public void propertyChange(PropertyChangeEvent event) {
                try {
                    setter.invoke(bean,
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
new Object[] { editor.getValue() });

} catch (IllegalAccessException exception) {
}

} catch (InvocationTargetException exception) {

} return editor;

} catch (InstantiationException exception) {
    return null;
}

} catch (IllegalAccessException exception) {
    return null;
}

} catch (InvocationTargetException exception) {
    return null;
}

} */

/** Wraps a property editor into a component.
   * @param editor the editor to wrap
   * @return a button (if there is a custom editor),
   * combo box (if the editor has tags), or text field (otherwise)
   */

public Component getEditorComponent(final PropertyEditor editor) {
    String[] tags = editor.getTags();
    String text = editor.getAsText();
    if (editor.supportsCustomEditor()) {
        // Make a button that pops up the custom editor
        final JButton button = new JButton();
        // if the editor is paintable, have it paint an icon
if (editor.isPaintable())
{
    button.setIcon(new Icon()
    {
        public int getIconWidth() { return WIDTH - 8; }
        public int getIconHeight() { return HEIGHT - 8; }
        
        public void paintIcon(Component c, Graphics g, int x, int y)
        {
            g.translate(x, y);
            Rectangle r = new Rectangle(0, 0,
                getIconWidth(), getIconHeight());
            Color oldColor = g.getColor();
            g.setColor(Color.BLACK);
            editor.paintValue(g, r);
            g.setColor(oldColor);
            g.translate(-x, -y);
        }
    });
    
    else
    {
        button.setText(buttonText(text));
        // pop up custom editor when button is clicked
        button.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                JOptionPane.showMessageDialog(null,
                    editor.getCustomEditor());
                if (editor.isPaintable())
                    button.repaint();
                else
                    button.setText(buttonText(editor.getAsText()));
            }
        });
    }
}

else if (tags != null)
final JComboBox comboBox = new JComboBox(tags);
comboBox.setSelectedItem(text);
comboBox.addItemListener(new ItemListener()
{
    public void itemStateChanged(ItemEvent event)
    {
        if (event.getStateChange() == ItemEvent.SELECTED)
            editor.setAsText((String) comboBox.getSelectedItem());
    }
});
return comboBox;
else
{
    final JTextField textField = new JTextField(text, 10);
textField.getDocument().addDocumentListener(new DocumentListener()
{
    public void insertUpdate(DocumentEvent e)
    {
        try
        {
            editor.setAsText(textField.getText());
        }
        catch (IllegalArgumentException exception) {} // Catch exception
    }
    public void removeUpdate(DocumentEvent e)
    {
        try
        {
            editor.setAsText(textField.getText());
        }
        catch (IllegalArgumentException exception) {} // Catch exception
    }
});
}
public void changedUpdate(DocumentEvent e) {
    }
    return textField;
}

/**
 * Formats text for the button that pops up a custom editor.
 * @param text the property value as text
 * @return the text to put on the button
 */
private static String buttonText(String text) {
    if (text == null || text.equals(""))
        return " ";
    if (text.length() > MAX_TEXT_LENGTH)
        return text.substring(0, MAX_TEXT_LENGTH) + "...";
    return text;
}

private ArrayList changeListeners = new ArrayList();
private static final int WIDTH = 100;
private static final int HEIGHT = 25;
private static final int MAX_TEXT_LENGTH = 15;
import java.awt.*;
import javax.swing.*;

/**
 * This program tests the property sheet by displaying a slider and a property sheet to edit the slider's properties.
 */
public class PropertySheetTest
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        Component comp = new JSlider();
        frame.getContentPane().add(comp);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.show();
        JPanel editor = new PropertySheet(comp);
        JComponent pane = new JScrollPane(editor);
        pane.setPreferredSize(new Dimension(400, 300));
        JOptionPane.showMessageDialog(frame, pane);
    }
}