Frameworks

- 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:
  - Frameworks
  - Applets as a simple framework
  - The collections framework
  - A graph editor framework
  - Enhancing the graph editor framework
**Frameworks**

- Set of cooperating classes
- Structures the essential mechanisms of a problem domain
- Example: Swing is a GUI framework
- Framework != design pattern
- Typical framework uses multiple design patterns
Application Frameworks

- Implements services common to a type of applications
- Programmer forms subclasses of framework classes
- Result is an application
- Inversion of control: framework controls execution flow
Applets

- Applet: Java program that runs in a web browser
- Programmer forms subclass of Applet or JApplet
- Overwrites
  - init/destroy
  - start/stop
  - paint
Applets

Tom Gray (t.grey@iac.unl.edu) and Bradley A. Smith (baldar@home.com) have done some amazing work converting Jmol for use as an Applet. Here are two samples of the Jmol Applet with Caffeine (as2) and Methanol (CML):

Controls:
- FP: move the molecule through the molecular orientation
- XD and XD: expand and contract the molecule

previous | start | next .... [Slide 5] ....
Applets

- Interacts with ambient browser
  - `getParameter`
  - `showDocument`
- HTML page contains applet tag and parameters

```html
<applet code="BannerApplet.class" width="300" height="100">
  <param name="message" value="Hello, World!"/>
  <param name="fontname" value="Serif"/>
  <param name="fontsize" value="64"/>
  <param name="delay" value="10"/>
</applet>
```
Example Applet

- Shows scrolling banner
- init reads parameters
- start/stop start and stop timer
- paint paints the applet surface
- Ch8/applet/BannerApplet.java
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.*;

public class BannerApplet extends Applet
{
    public void init()
    {
        message = getParameter("message");
        String fontname = getParameter("fontname");
        int fontsize = Integer.parseInt(getParameter("fontsize"));
        delay = Integer.parseInt(getParameter("delay"));
        font = new Font(fontname, Font.PLAIN, fontsize);
        Graphics2D g2 = (Graphics2D) getGraphics();
        FontRenderContext context = g2.getFontRenderContext();
        bounds = font.getStringBounds(message, context);

        timer = new Timer(delay, new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                start--;
                if (start + bounds.getWidth() < 0)
                {
                    start = getWidth();
                    repaint();
                }
            }
        });

        public void start()
        {
            timer.start();
        }

        public void stop()
        {
            timer.stop();
        }
}
public void paint(Graphics g) {
    g.setFont(font);
    g.drawString(message, start, (int)-bounds.getY());
}

private Timer timer;
private int start;
private int delay;
pribate String message;
private Font font;
private Rectangle2D bounds;
Example Applet

Hello, W
Applets as a Framework

- Applet programmer uses inheritance
- Applet class deals with generic behavior (browser interaction)
- Inversion of control: applet calls init, start, stop, destroy

previous | start | next .... [Slide 9] ....
Collections Framework

- Java library supplies standard data structures
- Supplies useful services (e.g. Collections.sort, Collections.shuffle)
- Framework: Programmers can supply additional data structures, services
- New data structures automatically work with services
- New services automatically work with data structures
Collections Framework: Interface Types

- **Collection**: the most general collection interface type
- **Set**: an unordered collection that does not permit duplicate elements
- **SortedSet**: a set whose elements are visited in sorted order
- **List**: an ordered collection
Collections Framework: Classes

- **HashSet**: a set implementation that uses hashing to locate the set elements
- **TreeSet**: a sorted set implementation that stores the elements in a balanced binary tree
- **LinkedList** and **ArrayList**: two implementations of the List interface type
Collections Framework

```
HashSet  arrow (interface) Collection
     \                  /
       v                  v
      Set                  Set
       \                  /
       v                  v
      SortedSet            List
                                 \                  /
                                 v                  v
                                 ArrayList           LinkedList
```

previous | start | next .... [Slide 13] ....
Collection Interface Type

- Collection holds elements in some way
- Different data structures have different storage strategies

boolean add(Object obj)
boolean addAll(Collection c)
void clear()
boolean contains(Object obj)
boolean containsAll(Collection c)
boolean equals(Object obj)
int hashCode()
boolean isEmpty()
Iterator iterator()
boolean remove(Object obj)
boolean removeAll(Collection c)
boolean retainAll(Collection c)
int size()
Object[] toArray()
Object[] toArray(Object[] a)
**Iterator Interface Type**

- Iterator traverses elements of collection

boolean hasNext()
Object next()
void remove()
AbstractCollection Class

- Collection is a hefty interface
- Convenient for clients, inconvenient for implementors
- Many methods can be implemented from others (Template method!)
- Example: toArray

```java
public Object[] toArray()
{
    Object[] result = new Object[size()];
    Iterator e = iterator();
    for (int i=0; e.hasNext(); i++)
        result[i] = e.next();
    return result;
}
```
AbstractCollection Class

- Can’t place template methods in interface
- Place them in AbstractCollection class
- AbstractCollection convenient superclass for implementors
- Only two methods undefined: size, iterator
Adding a new Class to the Framework

- Use queue from chapter 3
- Supply an iterator (with do-nothing `remove` method)
- `add` method always returns `true`
- Ch8/queue/Queue.java
- Ch8/queue/QueueTest.java
import java.util.*;

/**
 * A first-in, first-out bounded collection of objects.
 */
public class Queue extends AbstractCollection {

    /**
     * Constructs an empty queue.
     * @param capacity the maximum capacity of the queue
     * @precondition capacity > 0
     */
    public Queue(int capacity) {
        elements = new Object[capacity];
        count = 0;
        head = 0;
        tail = 0;
    }

    public Iterator iterator() {
        return new Iterator {
            public boolean hasNext() {
                return visited < count;
            }

            public Object next() {
                int index = (head + visited) % elements.length;
                Object r = elements[index];
                visited++;
                return r;
            }

            public void remove() {
                throw new UnsupportedOperationException();
            }
        };
    }
}
private int visited = 0;

/**
 * Remove object at head.
 * @return the object that has been removed from the queue
 * @precondition size() > 0
 */
public Object removeFirst()
{
    Object r = elements[head];
    head = (head + 1) % elements.length;
    count--;  
    return r;
}

/**
 * Append an object at tail.
 * @param anObject the object to be appended
 * @return true since this operation modifies the queue.
 * (This is a requirement of the collections framework.)
 * @precondition !isFull()
 */
public boolean add(Object anObject)
{
    elements[tail] = anObject;
    tail = (tail + 1) % elements.length;
    count++;
    return true;
}

/**
 * Checks whether this queue is full.
 */
```java
@return true if the queue is full
public boolean isFull()
{
    return count == elements.length;
}

/**
   * Gets object at head.
   * @return the object that is at the head of the queue
   * @pre condition size() > 0
   */
public Object getFirst()
{
    return elements[head];
}

private Object[] elements;
private int head;
private int tail;
private int count;
```
import java.util.*;

public class QueueTest
{
    public static void main(String[] args)
    {
        Queue q = new Queue(10);
        q.add("Belgium");
        q.add("Italy");
        q.add("France");
        q.removeFirst();
        q.add("Thailand");

        ArrayList a = new ArrayList();
        a.addAll(q);
        System.out.println("Result of bulk add: " + a);
        System.out.println("Minimum: " + Collections.min(q));
    }
}
Adding a new Class to the Framework

- `interface` Collection
- Abstract Collection
  - Queue
Sets

- Set interface adds no methods to Collection!
- Conceptually, sets are a subtype of collections
- Sets don’t store duplicates of the same element
- Sets are unordered
- Separate interface: an algorithm can require a Set
Lists

- Lists are *ordered*
- Each list position can be accessed by an integer index
- Subtype methods:

```java
boolean add(int index, Object obj)
boolean addAll(int index, Collection c)
Object get(int index)
int indexOf(Object obj)
int lastIndexOf(Object obj)
ListIterator listIterator()
ListIterator listIterator(int index)
Object remove(int index)
Object set(int index, int Object)
List subList(int fromIndex, int toIndex)
```
List Iterators

- Indexing
- Bidirectional behavior
- Subtype methods:

```java
int nextIndex()
int previousIndex()
boolean hasPrevious()
Object previous()
void set(Object obj)
```
List Classes

- ArrayList
- LinkedList
- Indexed access of linked list elements is possible, but slow
- Weakness in the design
- Partial fix in Java 1.4: RandomAccess interface
Optional Operations

- Many operations tagged as "optional"
- Example: `Collection.add`, `Collection.remove`
- Default implementation throws exception
- Why have optional operations?
Views

- View = collection that shows objects that are stored elsewhere
- Example: Arrays.asList
- String[] strings = { "Kenya", "Thailand", "Portugal" };
  List view = Arrays.asList(strings)
- Does not copy elements!
- Can use view for common services
  otherList.addAll(view);
Views

- `get/set` are defined to access underlying array
- `Arrays.asList` view has no `add/remove` operations
- Can’t grow/shrink underlying array
- Several kinds of views:
  - read-only
  - modifyable
  - resizable
  ...
- Optional operations avoid inflation of interfaces
- Controversial design decision
Graph Editor Framework

- Problem domain: interactive editing of diagrams
- Graph consists of nodes and edges
- Class diagram:
  nodes are rectangles
  edges are arrows
- Electronic circuit diagram:
  nodes are transistors, resistors
  edges are wires
Graph Editor Framework

- Traditional approach: programmer starts from scratch for every editor type
- Framework approach: Programmer extends graph, node, edge classes
- Framework handles UI, load/save, ...
- Our framework is kept simple
- Violet uses extension of this framework
User Interface

- Toolbar on top
- Grabber button for selecting nodes/edges
- Buttons for current node/edge type
- Menu
- Drawing area
Mouse Operations

- Click on empty space: current node inserted
- Click on node or edge: select it
- Drag node when current tool an edge: connect nodes
- Drag node when current tool not an edge: move node
Division of Responsibility

- Divide code between
  - framework
  - specific application
- Rendering is app specific (e.g. transistor)
- Hit testing is app specific (odd node shapes)
- Framework draws toolbar
- Framework does mouse listening
Adding Nodes and Edges

- Framework draws toolbar
- How does it know what nodes/edges to draw?
- App gives a list of nodes/edges to framework at startup
- How does app specify nodes/edges?
  - Class names? ("Transistor")
  - Class objects? (Transistor.class)
  - Node, Edge objects? (new Transistor())
Adding Nodes and Edges

- Objects are more flexible than classes
- new CircleNode(Color.BLACK)
  new CircleNode(Color.WHITE)
- When user inserts new node, the toolbar node is cloned
  Node prototype = node of currently selected toolbar button;
  Node newNode = (Node) prototype.clone();
  Point2D mousePoint = current mouse position;
  graph.add(newNode, mousePoint);
- Example of PROTOTYPE pattern
PROTOTYPE Pattern

Context

1. A system instantiates objects of classes that are not known when the system is built.
2. You do not want to require a separate class for each kind of object.
3. You want to avoid a separate hierarchy of classes whose responsibility it is to create the objects.

Solution

1. Define a prototype interface type that is common to all created objects.
2. Supply a prototype object for each kind of object that the system creates.
3. Clone the prototype object whenever a new object of the given kind is required.
PROTOTYPE Pattern

Creator

createInstance()

«interface» Prototype

Clones the prototype

Concrete Prototype1

Concrete Prototype2

previous | start | next .... [Slide 37] ....
## PROTOTYPE Pattern

<table>
<thead>
<tr>
<th>Name in Design Pattern</th>
<th>Actual name (graph editor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype</td>
<td>Node</td>
</tr>
<tr>
<td>ConcretePrototype1</td>
<td>CircleNode</td>
</tr>
<tr>
<td>Creator</td>
<td>The GraphPanel that handles the mouse operation for adding new nodes</td>
</tr>
</tbody>
</table>

previous | start | next .... [Slide 38] ....
Framework Classes

- Framework programmer implements Node/Edge interfaces
- draw draws node/edge
- getBounds returns enclosing rectangle (to compute total graph size for scrolling)
- Edge.getStart, getEnd yield start/end nodes
- Node.getConnectionPoint computes attachment point on shape boundary
- Edge.getConnectionPoints yields start/end coordinates (for grabbers)
- clone overridden to be public
Node Connection Points

Center of node

Boundary point

Exterior point
Framework Classes

- AbstractEdge class for convenience
- Programmer implements Node/Edge type or extends AbstractEdge
- Ch8/graphed/Node.java
- Ch8/graphed/Edge.java
- Ch8/graphed/AbstractEdge.java
import java.awt.*;
import java.awt.geom.*;
import java.io.*;

/**
 * A node in a graph.
 */
public interface Node extends Serializable, Cloneable {
    /**
     * Draw the node.
     * @param g2 the graphics context
     */
    void draw(Graphics2D g2);

    /**
     * Translates the node by a given amount.
     * @param dx the amount to translate in the x-direction
     * @param dy the amount to translate in the y-direction
     */
    void translate(double dx, double dy);

    /**
     * Tests whether the node contains a point.
     * @param aPoint the point to test
     * @return true if this node contains aPoint
     */
    boolean contains(Point2D aPoint);

    /**
     * Get the best connection point to connect this node with another node. This should be a point on the boundary of the shape of this node.
     * @param aPoint an exterior point that is to be joined with this node
     * @return the recommended connection point
     */
    Point2D getConnectionPoint(Point2D aPoint);

    /**
     * Get the bounding rectangle of the shape of this node
     */
@return the bounding rectangle
*/
Rectangle2D getBounds();
Object clone();
}
import java.awt.*;
import java.awt.geom.*;
import java.io.*;

/**
 * An edge in a graph.
 */
public interface Edge extends Serializable, Cloneable {

    /**
     * Draw the edge.
     * @param g2 the graphics context
     */
    void draw(Graphics2D g2);

    /**
     * Tests whether the edge contains a point.
     * @param aPoint the point to test
     * @return true if this edge contains aPoint
     */
    boolean contains(Point2D aPoint);

    /**
     * Connects this edge to two nodes.
     * @param aStart the starting node
     * @param anEnd the ending node
     */
    void connect(Node aStart, Node anEnd);

    /**
     * Gets the starting node.
     * @return the starting node
     */
    Node getStart();

    /**
     * Gets the ending node.
     * @return the ending node
     */
    Node getEnd();
}
/**
 * Gets the points at which this edge is connected to its nodes.
 * @return a line joining the two connection points
 */
    Line2D getConnectionPoints();

/**
 * Gets the smallest rectangle that bounds this edge.
 * The bounding rectangle contains all labels.
 * @return the bounding rectangle
 */
    Rectangle2D getBounds(Graphics2D g2);

    Object clone();
}
import java.awt.*;
import java.awt.geom.*;

/**
 * A class that supplies convenience implementations for a number of methods in the Edge interface type.
 */
public abstract class AbstractEdge implements Edge {

    public Object clone()
    {
        try {
            return super.clone();
        } catch (CloneNotSupportedException exception) {
            return null;
        }
    }

    public void connect(Node s, Node e)
    {
        start = s;
        end = e;
    }

    public Node getStart()
    {
        return start;
    }

    public Node getEnd()
    {
        return end;
    }

    public Rectangle2D getBounds(Graphics2D g2)
    {
        Line2D conn = getConnectionPoints();
        Rectangle2D r = new Rectangle2D.Double();
    }
42: r.setFrameFromDiagonal(conn.getX1(), conn.getY1(),
43:  conn.getX2(), conn.getY2());
44: return r;
45: }
46:
47: public Line2D getConnectionPoints()
48: {
49:  Rectangle2D startBounds = start.getBounds();
50:  Rectangle2D endBounds = end.getBounds();
51:  Point2D startCenter = new Point2D.Double(
52:     startBounds.getCenterX(), startBounds.getCenterY());
53:  Point2D endCenter = new Point2D.Double(
54:     endBounds.getCenterX(), endBounds.getCenterY());
55:  return new Line2D.Double(
56:     start.getConnectionPoint(endCenter),
57:     end.getConnectionPoint(startCenter));
58: }
59:
60: private Node start;
61: private Node end;
Framework Classes

- Graph collects nodes and edges
- Subclasses override methods
  ```java
  public abstract Node[] getNodePrototypes()
  public abstract Edge[] getEdgePrototypes()
  ```
- Ch8/graphed/Graph.java
import java.awt.*;
import java.awt.geom.*;
import java.io.*;
import java.util.*;
import java.util.List;

/**
  A graph consisting of selectable nodes and edges.
*/
public abstract class Graph implements Serializable {

  /**
   Constructs a graph with no nodes or edges.
   */
  public Graph() {
    nodes = new ArrayList();
    edges = new ArrayList();
  }

  /**
   Adds an edge to the graph that joins the nodes containing
   the given points. If the points aren’t both inside nodes,
   then no edge is added.
   @param e the edge to add
   @param p1 a point in the starting node
   @param p2 a point in the ending node
   */
  public boolean connect(Edge e, Point2D p1, Point2D p2) {
    Node n1 = findNode(p1);
    Node n2 = findNode(p2);
    if (n1 != null && n2 != null) {
      e.connect(n1, n2);
      edges.add(e);
      return true;
    }
    return false;
  }
}
public boolean add(Node n, Point2D p) {
    Rectangle2D bounds = n.getBounds();
    n.translate(p.getX() - bounds.getX(), p.getY() - bounds.getY());
    nodes.add(n);
    return true;
}

public Node findNode(Point2D p) {
    for (int i = nodes.size() - 1; i >= 0; i--)
    {
        Node n = (Node) nodes.get(i);
        if (n.contains(p)) return n;
    }
    return null;
}

public Edge findEdge(Point2D p) {
    for (int i = edges.size() - 1; i >= 0; i--)
    {
        Edge e = (Edge) edges.get(i);
        if (e.contains(p)) return e;
/**
 * Draws the graph
 * @param g2 the graphics context
 */

class Graph {
    public void draw(Graphics2D g2) {
        for (int i = 0; i < nodes.size(); i++) {
            Node n = (Node) nodes.get(i);
            n.draw(g2);
        }
    }

    for (int i = 0; i < edges.size(); i++) {
        Edge e = (Edge) edges.get(i);
        e.draw(g2);
    }
}

/**
 * Removes a node and all edges that start or end with that node
 * @param n the node to remove
 */

class Graph {
    public void removeNode(Node n) {
        for (int i = edges.size() - 1; i >= 0; i--) {
            Edge e = (Edge) edges.get(i);
            if (e.getStart() == n || e.getEnd() == n)
                edges.remove(e);
        }
    }
}

/**
 * Removes an edge from the graph.
 * @param e the edge to remove
 */

class Graph {
    public void removeEdge(Edge e) {
    }
}
public void removeEdge(Edge e) {
    edges.remove(e);
}
/**
 * Gets the smallest rectangle enclosing the graph
 * @param g2 the graphics context
 * @return the bounding rectangle
 */
public Rectangle2D getBounds(Graphics2D g2) {
    Rectangle2D r = null;
    for (int i = 0; i < nodes.size(); i++) {
        Node n = (Node) nodes.get(i);
        Rectangle2D b = n.getBounds();
        if (r == null) r = b;
        else r.add(b);
    }
    for (int i = 0; i < edges.size(); i++) {
        Edge e = (Edge) edges.get(i);
        r.add(e.getBounds(g2));
    }
    return r == null ? new Rectangle2D.Double() : r;
}
/**
 * Gets the node types of a particular graph type.
 * @return an array of node prototypes
 */
public abstract Node[] getNodePrototypes();
/**
 * Gets the edge types of a particular graph type.
 * @return an array of edge prototypes
 */
public abstract Edge[] getEdgePrototypes();
/**
   * Gets the nodes of this graph.
   * @return an unmodifiable list of the nodes
   */
   public List getNodes()
   {
       return Collections.unmodifiableList(nodes);
    }

/**
   * Gets the edges of this graph.
   * @return an unmodifiable list of the edges
   */
   public List getEdges()
   {
       return Collections.unmodifiableList(edges);
    }

private ArrayList nodes;
private ArrayList edges;
Framework UI Classes

- **GraphFrame**: a frame that manages the toolbar, the menu bar, and the graph panel.
- **ToolBar**: a panel that holds toggle buttons for the node and edge icons.
- **GraphPanel**: a panel that shows the graph and handles the mouse clicks and drags for the editing commands.
- Application programmers need not subclass these classes
A Framework Instance

- Simple application
- Draw black and white nodes
- Join nodes with straight lines
Programmer responsibilities

- For each node and edge type, define a class that implements the Node or Edge interface type
- Supply all required methods, such as drawing and containment testing.
- Define a subclass of the Graph class and supply getNodePrototypes, getEdgePrototypes
- Supply a class with a main method
A Framework Instance
A Framework Instance

- Ch8/graphed/SimpleGraph.java
- Ch8/graphed/SimpleGraphEditor.java
- Ch8/graphed/CircleNode.java
- Ch8/graphed/LineEdge.java
public class SimpleGraph extends Graph {

    public Node[] getNodePrototypes() {
        Node[] nodeTypes = {
            new CircleNode(Color.BLACK),
            new CircleNode(Color.WHITE)
        };
        return nodeTypes;
    }

    public Edge[] getEdgePrototypes() {
        Edge[] edgeTypes = {
            new LineEdge()
        };
        return edgeTypes;
    }
}
import javax.swing.*;
/**
 * A program for editing UML diagrams.
 */
public class SimpleGraphEditor
{
    public static void main(String[] args)
    {
        JFrame frame = new GraphFrame(new SimpleGraph());
        frame.show();
    }
}
```java
import java.awt.*;
import java.awt.geom.*;

/**
 * A circular node that is filled with a color.
 */
public class CircleNode implements Node {

    public CircleNode(Color aColor) {
        size = DEFAULT_SIZE;
        x = 0;
        y = 0;
        color = aColor;
    }

    public Object clone() {
        try {
            return super.clone();
        } catch (CloneNotSupportedException exception) {
            return null;
        }
    }

    public void draw(Graphics2D g2) {
        Ellipse2D circle = new Ellipse2D.Double(
                x, y, size, size);
        Color oldColor = g2.getColor();
        g2.setColor(color);
        g2.fill(circle);
        g2.setColor(oldColor);
        g2.draw(circle);
    }
}
```
public void translate(double dx, double dy) {
    x += dx;
    y += dy;
}

public boolean contains(Point2D p) {
    Ellipse2D circle = new Ellipse2D.Double(x, y, size, size);
    return circle.contains(p);
}

public Rectangle2D getBounds() {
    return new Rectangle2D.Double(x, y, size, size);
}

public Point2D getConnectionPoint(Point2D other) {
    double centerX = x + size / 2;
    double centerY = y + size / 2;
    double dx = other.getX() - centerX;
    double dy = other.getY() - centerY;
    double distance = Math.sqrt(dx * dx + dy * dy);
    if (distance == 0) return other;
    else return new Point2D.Double(
        centerX + dx * (size / 2) / distance,
        centerY + dy * (size / 2) / distance);
}

private int x;
private int y;
private int size;
private Color color;
private static final int DEFAULT_SIZE = 20;
import java.awt.*;
import java.awt.geom.*;

/**
   * An edge that is shaped like a straight line.
   */
public class LineEdge extends AbstractEdge {
    public void draw(Graphics2D g2) {
        g2.draw(getConnectionPoints());
    }
    public boolean contains(Point2D aPoint) {
        final double MAX_DIST = 2;
        return getConnectionPoints().ptSegDist(aPoint) < MAX_DIST;
    }
}
Generic Framework Code

• Framework frees application programmer from tedious programming
• Framework can do significant work without knowing node/edge types
• Analyze two scenarios
  • Add new node
  • Add new edge
Add New Node

public void mousePressed(MouseEvent event)
{
    Point2D mousePoint = event.getPoint();
    Object tool = toolBar.getSelectedTool();
    ...
    if (tool instanceof Node)
    {
        Node prototype = (Node) tool;
        Node newNode = (Node)prototype.clone();
        graph.add(newNode, mousePoint);
    }
    ...
    repaint();
}
Add New Node

mouse listener

: Mouse Event

:ToolBar

prototype : Node

:Graph

generate

generateSelectedTool

clone

add

previous | start | next .... [Slide 50] ....
Add New Edge

- First check if mouse was pressed inside existing node

```java
public Node findNode(Point2D p)
{
    for (int i = 0; i < nodes.size(); i++)
    {
        Node n = (Node) nodes.get(i);
        if (n.contains(p)) return n;
    }
    return null;
}
```

Add New Edge

- mousePressed:
  - Check if mouse point inside node
  - Check if current tool is edge
  - Mouse point is start of rubber band
- mouseDragged:
  - Mouse point is end of rubber band; repaint
- mouseReleased:
  - Add edge to graph
Add New Edge
Enhancing the Framework

- Edit node/edge properties
  - Node colors
  - Edge styles (solid/dotted)
- Framework enhancement: Edit->Properties menu pops up property dialog
Enhancing the Framework
Enhancing the Framework

- How to implement the dialog?
- Solved in chapter 7--bean properties!
- CircleNode exposes color property:
  
  ```java
  Color getColor()
  void setColor(Color newValue)
  ```
- Property editor automatically edits color!
Using the Framework Enhancement

- Add dotted lines
- Define enumerated type LineStyle
- Two instances LineStyle.SOLID, LineStyle.DOTTED
- Add lineStyle property to LineEdge
- LineStyle has method getStroke()
- LineEdge.draw calls getStroke()
- Supply property editor for LineStyle type
- Property editor now edits line style!
Another Framework Instance

- UML Class diagram editor
- "Violet lite"
Another Framework Instance

- RectangularNode
- SegmentedLineEdge
- GeneralPathEdge uses general path for containment testing
- ArrowHead, BentStyle enumerate arrow and line styles
- MultiLineString property for class compartments
- ClassNode, ClassRelationshipEdge, ClassDiagramGraph
- No change to basic framework!
Edge Properties
Multiline String Property Editor

previous | start | next .... [Slide 62] ....
Enhancing the Framework II

- Violet is based on an enhancement of the book’s framework
- Adds many options
- 
  - graphics export
  - grid
  - multiple windows
- Can add 3 simple graph editor classes to that framework
- App tracks framework evolution at no cost to app programmer
Enhancing the Framework II