COMP 303 - Lecture Notes for Week 8 - 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

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

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

previous | start | next .... [Slide 2] ....

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

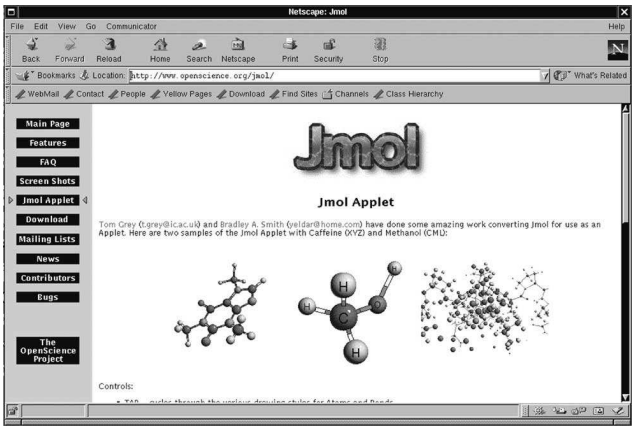
previous | start | next .... [Slide 3] ....

Applets

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

previous | start | next .... [Slide 4] ....

Applets



Applets

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

```
<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


```
01: import java.applet.*;
02: import java.awt.*;
03: import java.awt.event.*;
04: import java.awt.font.*;
05: import java.awt.geom.*;
06: import javax.swing.*;
07:
08: public class BannerApplet extends Applet
09: {
10:    public void init()
11:    {
12:       message = getParameter("message");
13:       String fontname = getParameter("fontname");
14:       int fontsize = Integer.parseInt(getParameter("fontsize"));
15:       delay = Integer.parseInt(getParameter("delay"));
16:       font = new Font(fontname, Font.PLAIN, fontsize);
17:       Graphics2D g2 = (Graphics2D) getGraphics();
18:       FontRenderContext context = g2.getFontRenderContext();
19:       bounds = font.getStringBounds(message, context);
20:
21:       timer = new Timer(delay, new
22:          ActionListener()
23:          {
24:             public void actionPerformed(ActionEvent event)
25:             {
26:                start--;
27:                if (start + bounds.getWidth() < 0)
28:                   start = getWidth();
29:                repaint();
30:             }
31:          });
32:    }
33:
34:    public void start()
35:    {
36:       timer.start();
37:    }
38:
39:    public void stop()
40:    {
41:       timer.stop();
```

```
42:    }
43:
44:    public void paint(Graphics g)
45:    {
46:       g.setFont(font);
47:       g.drawString(message, start, (int)-bounds.getY());
48:    }
49:
50:    private Timer timer;
51:    private int start;
52:    private int delay;
53:    private String message;
54:    private Font font;
55:    private Rectangle2D bounds;
56: }
```

Example Applet

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`

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

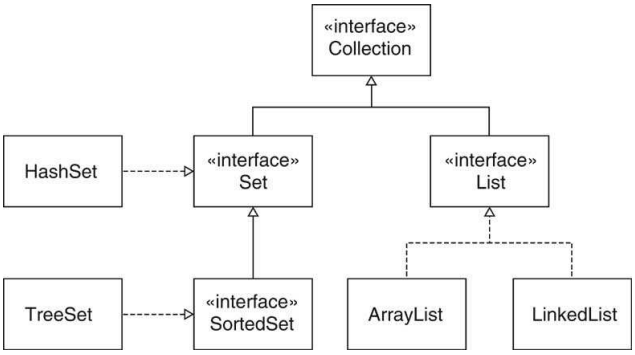
Collections Framework: Interface Types

- `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



`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, inconvenientfor implementors
- Many methods can be implemented from others (Template method!)
- Example: `toArray`

```
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`

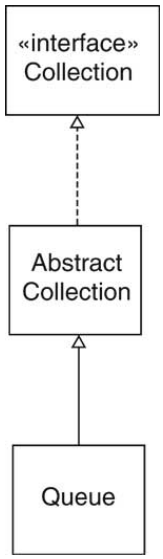
Adding a new Class to the Framework

- 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

```
001: import java.util.*;
002:
003: /**
004:    A first-in, first-out bounded collection of objects.
005: */
006: public class Queue extends AbstractCollection
007: {
008:    /**
009:       Constructs an empty queue.
010:       @param capacity the maximum capacity of the queue
011:       @precondition capacity > 0
012:    */
013:    public Queue(int capacity)
014:    {
015:       elements = new Object[capacity];
016:       count = 0;
017:       head = 0;
018:       tail = 0;
019:    }
020:
021:    public Iterator iterator()
022:    {
023:       return new
024:          Iterator()
025:          {
026:             public boolean hasNext()
027:             {
028:                return visited < count;
029:             }
030:
031:             public Object next()
032:             {
033:                int index = (head + visited) % elements.length;
034:                Object r = elements[index];
035:                visited++;
036:                return r;
037:             }
038:
039:             public void remove()
040:             {
041:                throw new UnsupportedOperationException();
042:             }
043:
044:             private int visited = 0;
045:          };
046:    }
047:
048:    /**
049:       Remove object at head.
050:       @return the object that has been removed from the queue
051:       @precondition size() > 0
052:    */
053:    public Object removeFirst()
054:    {
055:       Object r = elements[head];
056:       head = (head + 1) % elements.length;
057:       count--;
058:       return r;
059:    }
060:
061:    /**
062:       Append an object at tail.
063:       @param anObject the object to be appended
064:       @return true since this operation modifies the queue.
065:       (This is a requirement of the collections framework.)
066:       @precondition !isFull()
067:    */
068:    public boolean add(Object anObject)
069:    {
070:       elements[tail] = anObject;
071:       tail = (tail + 1) % elements.length;
072:       count++;
073:       return true;
074:    }
075:
076:    public int size()
077:    {
078:       return count;
079:    }
080:
081:    /**
082:       Checks whether this queue is full.
083:       @return true if the queue is full
084:    */
085:    public boolean isFull()
086:    {
087:       return count == elements.length;
088:    }
089:
090:    /**
091:       Gets object at head.
092:       @return the object that is at the head of the queue
093:       @precondition size() > 0
094:    */
095:    public Object getFirst()
096:    {
097:       return elements[head];
098:    }
099:
100:    private Object[] elements;
101:    private int head;
102:    private int tail;
103:    private int count;
104: }
```

```
01: import java.util.*;
02:
03: public class QueueTest
04: {
05:    public static void main(String[] args)
06:    {
07:       Queue q = new Queue(10);
08:
09:       q.add("Belgium");
10:       q.add("Italy");
11:       q.add("France");
12:       q.removeFirst();
13:       q.add("Thailand");
14:
15:       ArrayList a = new ArrayList();
16:       a.addAll(q);
17:       System.out.println("Result of bulk add: " + a);
18:       System.out.println("Minimum: " + Collections.min(q));
19:    }
20: }
```

Adding a new Class to the Framework



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:

```
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

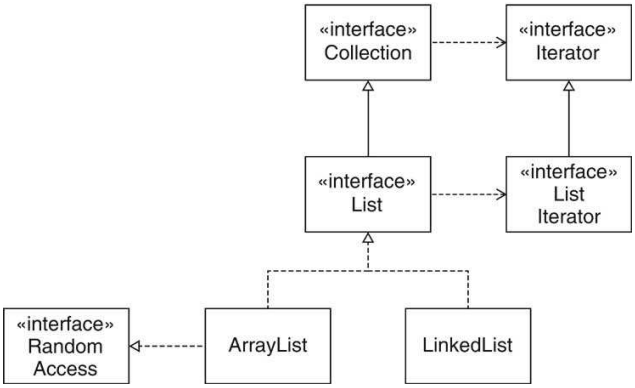
- Indexing
- Bidirectional behavior
- Subtype methods:

```
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

List Classes



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

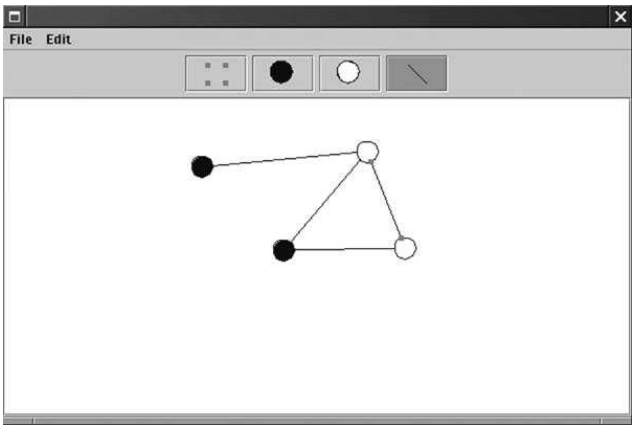
Graph Editor Framework

- 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

User Interface



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

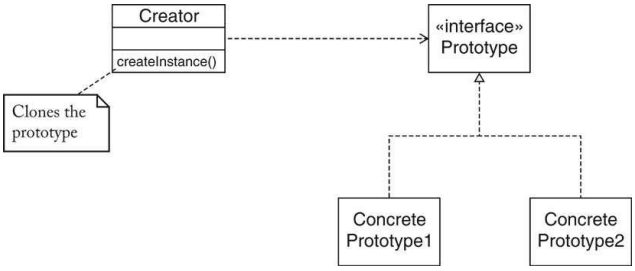
Context

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



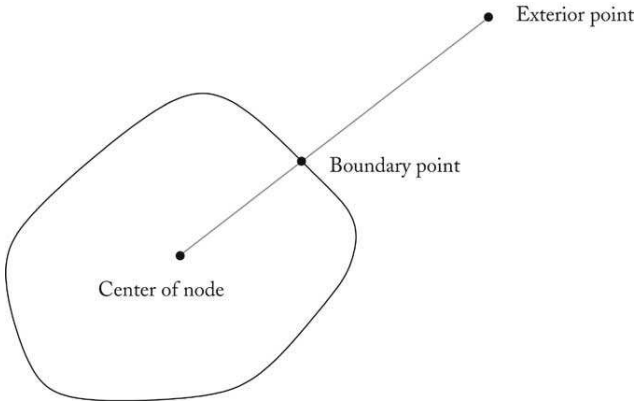
PROTOTYPE Pattern

Name in Design Pattern	Actual name (graph editor)
Prototype	Node
ConcretePrototype1	CircleNode
Creator	The `GraphPanel` that handles the mouse operation for adding new nodes

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



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


```
01: import java.awt.*;
02: import java.awt.geom.*;
03: import java.io.*;
04:
05: /**
06:    A node in a graph.
07: */
08: public interface Node extends Serializable, Cloneable
09: {
10:    /**
11:       Draw the node.
12:       @param g2 the graphics context
13:    */
14:    void draw(Graphics2D g2);
15:
16:    /**
17:       Translates the node by a given amount.
18:       @param dx the amount to translate in the x-direction
19:       @param dy the amount to translate in the y-direction
20:    */
21:    void translate(double dx, double dy);
22:
23:    /**
24:       Tests whether the node contains a point.
25:       @param aPoint the point to test
26:       @return true if this node contains aPoint
27:    */
28:    boolean contains(Point2D aPoint);
29:
30:    /**
31:       Get the best connection point to connect this node
32:       with another node. This should be a point on the boundary
33:       of the shape of this node.
34:       @param aPoint an exterior point that is to be joined
35:       with this node
36:       @return the recommended connection point
37:    */
38:    Point2D getConnectionPoint(Point2D aPoint);
39:
40:    /**
41:       Get the bounding rectangle of the shape of this node
```

```
42:       @return the bounding rectangle
43:    */
44:    Rectangle2D getBounds();
45:
46:    Object clone();
47: }
```

```
01: import java.awt.*;
02: import java.awt.geom.*;
03: import java.io.*;
04:
05: /**
06:    An edge in a graph.
07: */
08: public interface Edge extends Serializable, Cloneable
09: {
10:    /**
11:       Draw the edge.
12:       @param g2 the graphics context
13:    */
14:    void draw(Graphics2D g2);
15:
16:    /**
17:       Tests whether the edge contains a point.
18:       @param aPoint the point to test
19:       @return true if this edge contains aPoint
20:    */
21:    boolean contains(Point2D aPoint);
22:
23:    /**
24:       Connects this edge to two nodes.
25:       @param aStart the starting node
26:       @param anEnd the ending node
27:    */
28:    void connect(Node aStart, Node anEnd);
29:
30:    /**
31:       Gets the starting node.
32:       @return the starting node
33:    */
34:    Node getStart();
35:
36:    /**
37:       Gets the ending node.
38:       @return the ending node
39:    */
40:    Node getEnd();
41:
```

```
42:    /**
43:       Gets the points at which this edge is connected to
44:       its nodes.
45:       @return a line joining the two connection points
46:    */
47:    Line2D getConnectionPoints();
48:
49:    /**
50:       Gets the smallest rectangle that bounds this edge.
51:       The bounding rectangle contains all labels.
52:       @return the bounding rectangle
53:    */
54:    Rectangle2D getBounds(Graphics2D g2);
55:
56:    Object clone();
57: }
58:
```

```
01: import java.awt.*;
02: import java.awt.geom.*;
03:
04: /**
05:    A class that supplies convenience implementations for
06:    a number of methods in the Edge interface type.
07: */
08: public abstract class AbstractEdge implements Edge
09: {
10:    public Object clone()
11:    {
12:       try
13:       {
14:          return super.clone();
15:       }
16:       catch (CloneNotSupportedException exception)
17:       {
18:          return null;
19:       }
20:    }
21:
22:    public void connect(Node s, Node e)
23:    {
24:       start = s;
25:       end = e;
26:    }
27:
28:    public Node getStart()
29:    {
30:       return start;
31:    }
32:
33:    public Node getEnd()
34:    {
35:       return end;
36:    }
37:
38:    public Rectangle2D getBounds(Graphics2D g2)
39:    {
40:       Line2D conn = getConnectionPoints();
41:       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;
62: }
```

Framework Classes

- `Graph` collects nodes and edges
- Subclasses override methods
  `public abstract Node[] getNodePrototypes()`
  `public abstract Edge[] getEdgePrototypes()`
- Ch8/graphed/Graph.java


```
001: import java.awt.*;
002: import java.awt.geom.*;
003: import java.io.*;
004: import java.util.*;
005: import java.util.List;
006:
007: /**
008:    A graph consisting of selectable nodes and edges.
009: */
010: public abstract class Graph implements Serializable
011: {
012:    /**
013:       Constructs a graph with no nodes or edges.
014:    */
015:    public Graph()
016:    {
017:       nodes = new ArrayList();
018:       edges = new ArrayList();
019:    }
020:
021:    /**
022:       Adds an edge to the graph that joins the nodes containing
023:       the given points. If the points aren't both inside nodes,
024:       then no edge is added.
025:       @param e the edge to add
026:       @param p1 a point in the starting node
027:       @param p2 a point in the ending node
028:    */
029:    public boolean connect(Edge e, Point2D p1, Point2D p2)
030:    {
031:       Node n1 = findNode(p1);
032:       Node n2 = findNode(p2);
033:       if (n1 != null && n2 != null)
034:       {
035:          e.connect(n1, n2);
036:          edges.add(e);
037:          return true;
038:       }
039:       return false;
040:    }
041:
```

```
042:    /**
043:       Adds a node to the graph so that the top left corner of
044:       the bounding rectangle is at the given point.
045:       @param n the node to add
046:       @param p the desired location
047:    */
048:    public boolean add(Node n, Point2D p)
049:    {
050:       Rectangle2D bounds = n.getBounds();
051:       n.translate(p.getX() - bounds.getX(),
052:          p.getY() - bounds.getY());
053:       nodes.add(n);
054:       return true;
055:    }
056:
057:    /**
058:       Finds a node containing the given point.
059:       @param p a point
060:       @return a node containing p or null if no nodes contain p
061:    */
062:    public Node findNode(Point2D p)
063:    {
064:       for (int i = nodes.size() - 1; i >= 0; i--)
065:       {
066:          Node n = (Node) nodes.get(i);
067:          if (n.contains(p)) return n;
068:       }
069:       return null;
070:    }
071:
072:    /**
073:       Finds an edge containing the given point.
074:       @param p a point
075:       @return an edge containing p or null if no edges contain p
076:    */
077:    public Edge findEdge(Point2D p)
078:    {
079:       for (int i = edges.size() - 1; i >= 0; i--)
080:       {
081:          Edge e = (Edge) edges.get(i);
082:          if (e.contains(p)) return e;
```

```
083:       }
084:       return null;
085:    }
086:
087:    /**
088:       Draws the graph
089:       @param g2 the graphics context
090:    */
091:    public void draw(Graphics2D g2)
092:    {
093:       for (int i = 0; i < nodes.size(); i++)
094:       {
095:          Node n = (Node) nodes.get(i);
096:          n.draw(g2);
097:       }
098:
099:       for (int i = 0; i < edges.size(); i++)
100:       {
101:          Edge e = (Edge) edges.get(i);
102:          e.draw(g2);
103:       }
104:    }
105:
106:    /**
107:       Removes a node and all edges that start or end with that node
108:       @param n the node to remove
109:    */
110:    public void removeNode(Node n)
111:    {
112:       for (int i = edges.size() - 1; i >= 0; i--)
113:       {
114:          Edge e = (Edge) edges.get(i);
115:          if (e.getStart() == n || e.getEnd() == n)
116:             edges.remove(e);
117:       }
118:       nodes.remove(n);
119:    }
120:
121:    /**
122:       Removes an edge from the graph.
123:       @param e the edge to remove
```

```
124:    */
125:    public void removeEdge(Edge e)
126:    {
127:       edges.remove(e);
128:    }
129:
130:    /**
131:       Gets the smallest rectangle enclosing the graph
132:       @param g2 the graphics context
133:       @return the bounding rectangle
134:    */
135:    public Rectangle2D getBounds(Graphics2D g2)
136:    {
137:       Rectangle2D r = null;
138:       for (int i = 0; i < nodes.size(); i++)
139:       {
140:          Node n = (Node) nodes.get(i);
141:          Rectangle2D b = n.getBounds();
142:          if (r == null) r = b;
143:          else r.add(b);
144:       }
145:       for (int i = 0; i < edges.size(); i++)
146:       {
147:          Edge e = (Edge) edges.get(i);
148:          r.add(e.getBounds(g2));
149:       }
150:       return r == null ? new Rectangle2D.Double() : r;
151:    }
152:
153:    /**
154:       Gets the node types of a particular graph type.
155:       @return an array of node prototypes
156:    */
157:    public abstract Node[] getNodePrototypes();
158:
159:    /**
160:       Gets the edge types of a particular graph type.
161:       @return an array of edge prototypes
162:    */
163:    public abstract Edge[] getEdgePrototypes();
164:
```

```
165:    /**
166:       Gets the nodes of this graph.
167:       @return an unmodifiable list of the nodes
168:    */
169:    public List getNodes()
170:    {
171:       return Collections.unmodifiableList(nodes); }
172:
173:    /**
174:       Gets the edges of this graph.
175:       @return an unmodifiable list of the edges
176:    */
177:    public List getEdges()
178:    {
179:       return Collections.unmodifiableList(edges);
180:    }
181:
182:    private ArrayList nodes;
183:    private ArrayList edges;
184: }
185:
186:
187:
188:
189:
```

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

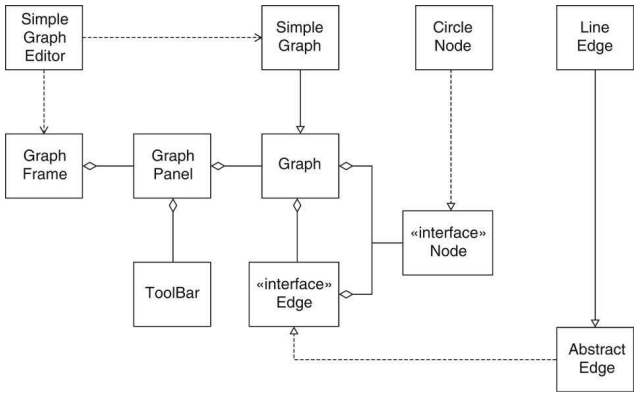
A Framework Instance

- 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

```
01: import java.awt.*;
02: import java.util.*;
03:
04: /**
05:    A simple graph with round nodes and straight edges.
06: */
07: public class SimpleGraph extends Graph
08: {
09:    public Node[] getNodePrototypes()
10:    {
11:       Node[] nodeTypes =
12:          {
13:             new CircleNode(Color.BLACK),
14:             new CircleNode(Color.WHITE)
15:          };
16:       return nodeTypes;
17:    }
18:
19:    public Edge[] getEdgePrototypes()
20:    {
21:       Edge[] edgeTypes =
22:          {
23:             new LineEdge()
24:          };
25:       return edgeTypes;
26:    }
27: }
28:
29:
30:
31:
32:
```

```
01: import javax.swing.*;
02:
03: /**
04:    A program for editing UML diagrams.
05: */
06: public class SimpleGraphEditor
07: {
08:    public static void main(String[] args)
09:    {
10:       JFrame frame = new GraphFrame(new SimpleGraph());
11:       frame.show();
12:    }
13: }
14:
```

```
01: import java.awt.*;
02: import java.awt.geom.*;
03:
04: /**
05:    A circular node that is filled with a color.
06: */
07: public class CircleNode implements Node
08: {
09:    /**
10:       Construct a circle node with a given size and color.
11:       @param aColor the fill color
12:    */
13:    public CircleNode(Color aColor)
14:    {
15:       size = DEFAULT_SIZE;
16:       x = 0;
17:       y = 0;
18:       color = aColor;
19:    }
20:
21:    public Object clone()
22:    {
23:       try
24:       {
25:          return super.clone();
26:       }
27:       catch (CloneNotSupportedException exception)
28:       {
29:          return null;
30:       }
31:    }
32:
33:    public void draw(Graphics2D g2)
34:    {
35:       Ellipse2D circle = new Ellipse2D.Double(
36:          x, y, size, size);
37:       Color oldColor = g2.getColor();
38:       g2.setColor(color);
39:       g2.fill(circle);
40:       g2.setColor(oldColor);
41:       g2.draw(circle);
```

```
42:    }
43:
44:    public void translate(double dx, double dy)
45:    {
46:       x += dx;
47:       y += dy;
48:    }
49:
50:    public boolean contains(Point2D p)
51:    {
52:       Ellipse2D circle = new Ellipse2D.Double(
53:          x, y, size, size);
54:       return circle.contains(p);
55:    }
56:
57:    public Rectangle2D getBounds()
58:    {
59:       return new Rectangle2D.Double(
60:          x, y, size, size);
61:    }
62:
63:    public Point2D getConnectionPoint(Point2D other)
64:    {
65:       double centerX = x + size / 2;
66:       double centerY = y + size / 2;
67:       double dx = other.getX() - centerX;
68:       double dy = other.getY() - centerY;
69:       double distance = Math.sqrt(dx * dx + dy * dy);
70:       if (distance == 0) return other;
71:       else return new Point2D.Double(
72:          centerX + dx * (size / 2) / distance,
73:          centerY + dy * (size / 2) / distance);
74:    }
75:
76:    private int x;
77:    private int y;
78:    private int size;
79:    private Color color;
80:    private static final int DEFAULT_SIZE = 20;
81: }
```

```
01: import java.awt.*;
02: import java.awt.geom.*;
03:
04: /**
05:    An edge that is shaped like a straight line.
06: */
07: public class LineEdge extends AbstractEdge
08: {
09:    public void draw(Graphics2D g2)
10:    {
11:       g2.draw(getConnectionPoints());
12:    }
13:
14:    public boolean contains(Point2D aPoint)
15:    {
16:       final double MAX_DIST = 2;
17:       return getConnectionPoints().ptSegDist(aPoint)
18:          < MAX_DIST;
19:    }
20: }
```

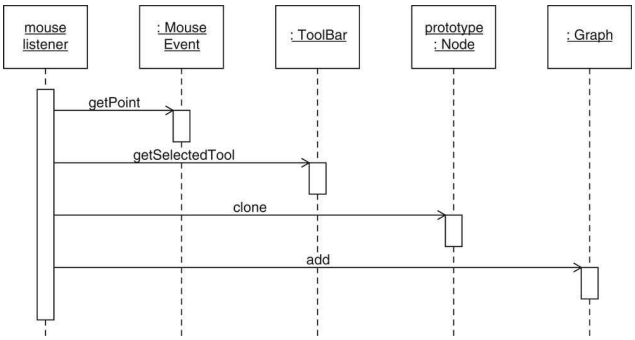
Generic Framework Code

- 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



Add New Edge

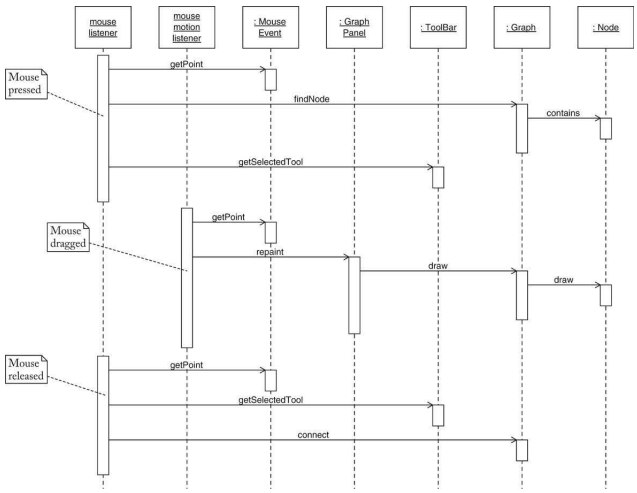
- First check if mouse was pressed inside existing node

```
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



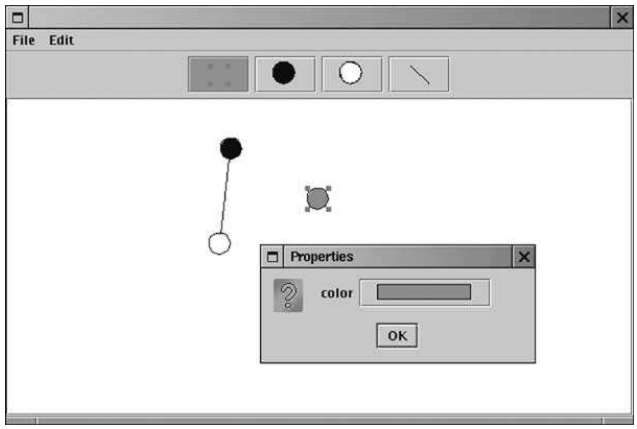
previous | start | next .... [Slide 54] ....

Enhancing the Framework

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

previous | start | next .... [Slide 55] ....

Enhancing the Framework



previous | start | next .... [Slide 56] ....

Enhancing the Framework

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

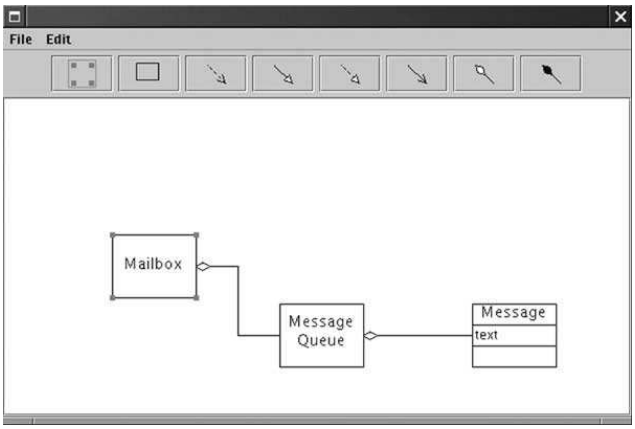
previous | start | next .... [Slide 57] ....

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

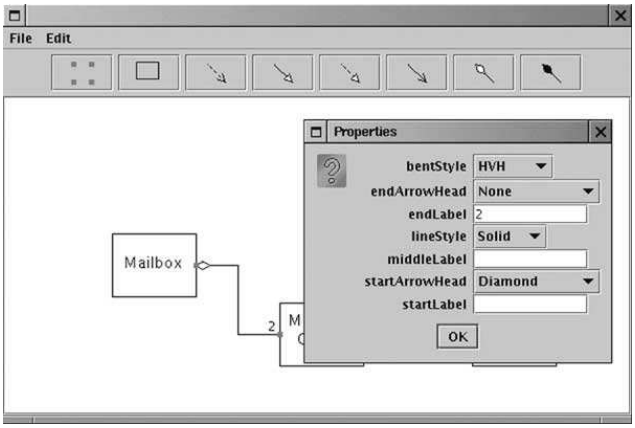
- UML Class diagram editor
- "Violet lite"



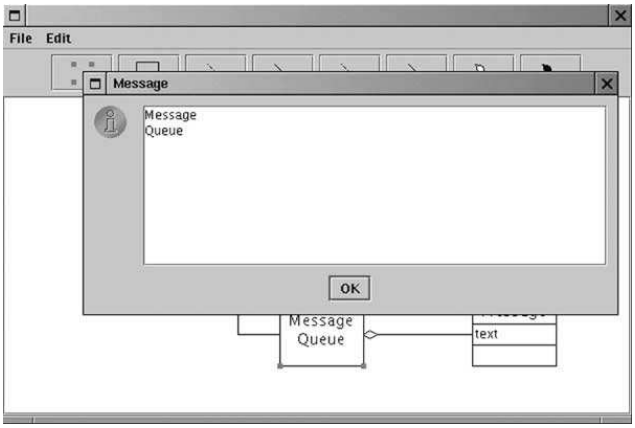
Another Framework Instance

- `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



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