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

- 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

```java
01: import java.awt.*;
02: import java.applet.*;
03: import java.awt.event.*;
04: import java.awt.font.*;
05: import java.awt.geom.*;
06: import javax.swing.*;
07: public class BannerApplet extends Applet
08: {
09:     public void init()
10:     {
11:         message = getParameter("message");
12:         String fontname = getParameter("fontname");
13:         int fontsize = Integer.parseInt(getParameter("fontsize"));
14:         delay = Integer.parseInt(getParameter("delay"));
15:         if (message == null) delay = 1000;
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:         timer = new Timer(delay, new ActionListener()
21:             {  
22:                 public void actionPerformed(ActionEvent event)
23:                 {
24:                     if (start > 0)
25:                     {
26:                         if (start + bounds.getWidth() < 0)
27:                         {
28:                             start = getWidth();
29:                             repaint();
30:                         }
31:                     }
32:                 }
33:             });
34:     }
35:     public void start()
36:     {
37:         timer.start();
38:     }
39:     public void stop()
40:     {
41:         timer.stop();
42:     }
```
42: }
43: public void paint(Graphics g)
44: {
45:     g.setFont(font);
46:     g.drawString(message, start, (int)bounds.getY());
47: }
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

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;
}
```

---

### 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
{
    private int visited = 0;
    private Object[] elements = new Object[10];
    int count = 0;
    int tail = 0;
    int head = 0;

    public Queue(int capacity)
    {
        if (capacity <= 0) throw new IllegalArgumentException();
        else {
            elements = new Object[capacity];
            count = 0;
            head = 0;
            tail = 0;
        }
    }

    public Object visit()
    {
        return visited;
    }

    public void remove()
    {
        throw new UnsupportedOperationException();
    }

    public void add(Object anObject)
    {
        if (count == 0)
            elements[tail] = anObject;
        else
            elements[tail] = anObject;
        count++;
        tail = (tail + 1) % elements.length;
    }

    public Object getFirst()
    {
        return elements[head];
    }

    public int size()
    {
        return count;
    }

    public boolean isFull()
    {
        return count == elements.length;
    }

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

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

    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 boolean removeFirst()
    {
        if (count == 0) return false;
        else {
            count--;
            return true;
        }
    }

    public void addAll(int... a)
    {
        for (int i = 0; i < a.length; i++)
            add(a[i]);
    }

    public void addAll(Object... a)
    {
        for (int i = 0; i < a.length; i++)
            add(a[i]);
    }

    public void addAll(ArrayList a)
    {
        for (int i = 0; i < a.size(); i++)
            add(a.get(i));
    }

    @precondition size() > 0
    @return the object that has been removed from the queue
    (This is a requirement of the collections framework.)
    @return true since this operation modifies the queue.
    @param anObject the object to be appended
    @precondition capacity > 0
    @param capacity the maximum capacity of the queue
}

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

        System.out.println("Result of bulk add: "+q);
        q.addAll("a");
        System.out.println("Result of bulk add: " + q);
    }
}
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() (extends 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

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

User Interface
- Toolbar on top
- Grabber button for selecting nodes/edges
- Buttons for current node/edge type
- Menu
- Drawing area
**User Interface**

![Image of a user interface with nodes and edges]

**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
  ```java
  Node prototype = node of currently selected toolbar button;
  Node newNode = (Node) prototype.clone();
  Point2D mousePoint = current mouse position; // assumes mouse Pressed
  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.

<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 GraphPanel1 that handles the mouse operation for adding new nodes</td>
</tr>
</tbody>
</table>

PROTOTYPE Pattern

Diagram showing the relationship between Creator, Prototype, and Concrete prototypes.
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

- Exterior point
- Boundary point
- Center of node

---

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`

---

```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:        */
27:        boolean contains(Point2D aPoint);
28:     
29:     /**
30:        Get the best connection point to connect this node
31:        with another node. This should be a point on the boundary
32:        of the shape of this node.
33:        @param aPoint an exterior point that is to be joined
34:        with this node
35:        */
36:        Point2D getConnectionPoint(Point2D aPoint);
37:     
38:     /**
39:        Get the bounding rectangle of the shape of this node
40: */
41: 
42:     
43: 
44: 
45: 
46: 
47: ```
import java.awt.*;
import java.awt.geom.*;
import java.io.*;

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

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

    Object clone();
}

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();
        return r;
    }
}
public abstract Edge[] getEdgePrototypes()

Ch8/graphed/Graph.java

Finds a node containing the given point.

@returns the edge containing p or null if no edges contain p

Edge e

Finds an edge containing the given point.

@params p a point
  p2 a point

Node n
Node n1
Node n2

Constructs a graph with no nodes or edges.

public abstract class Graph implements Serializable

Graph consists of selectable nodes and edges.

A graph consisting of selectable nodes and edges.

public abstract class Graph implements Serializable

previous | start | next ... [Slide 42] ...
/**
 * 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;

/**
 * Draws the graph
 *
 * @param g2 the graphics context
 */

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
     */

    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);
        }

        nodes.remove(n);
    }

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

    public void removeEdge(Edge e)
    {
        edges.remove(e);
    }

    /**
     * Gets the smallest rectangle enclosing the graph
     * @param g2 the graphics context
     */

    public Rectangle2D getBounds(Graphics2D g2)
    {
        if (nodes == null)
        return null;

        for (int i = 0; i < nodes.size(); i++)
        {
            Node n = (Node) nodes.get(i);
            Rectangle2D r = n.getBounds();
            if (r != null) r = r.union(r);
        }

        for (int i = 0; i < edges.size(); i++)
        {
            Edge e = (Edge) edges.get(i);
            r.add(e.getBounds());
        }

        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();
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

Ch8/graphed/SimpleGraph.java
Ch8/graphed/SimpleGraphEditor.java
Ch8/graphed/CircleNode.java
Ch8/graphed/LineEdge.java
import java.awt.*;
import java.awt.geom.*;

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

    private int x;
    private int y;
    private int size;
    private Color color;
    private static final int DEFAULT_SIZE = 20;

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

    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);
        g2.setColor(color);
        g2.fill(circle);
        g2.setColor(oldColor);
        g2.draw(circle);
    }
}

A simple graph with round nodes and straight edges.

public class SimpleGraphEditor

    {  
       JFrame frame = new GraphFrame(new SimpleGraph());
       frame.show();
    }
}

public class SimpleGraph

    {  
       Node[] nodeTypes;
       Edge[] edgeTypes;
       
       public Edge[] getEdgePrototypes()
       {  
           return edgeTypes;
       }

       public Node[] getNodePrototypes()
       {  
           return nodeTypes;
       }

       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);
       }
}

public class GraphFrame

    {  
       SimpleGraph graph = new SimpleGraph();
       
       public void translate(double dx, double dy)
       {  
           x += dx;
           y += dy;
       }

       public void main(String[] args)
       {  
           JFrame frame = new GraphFrame(new SimpleGraph());
           frame.show();
       }
}
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;
    }
}
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
- **mouseDragged:**
  - Mouse point is start of rubber band
- **mouseReleased:**
  - Mouse point is end of rubber band; repaint
  - Add edge to graph
Enhancing the Framework

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

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!
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

Multiline String Property Editor

![Multiline String Property Editor](image-url)