Object Oriented Programming and Design in Java

Session 16

Instructor: Bert Huang

Announcements

- Homework 3 out. Due **Monday**, Apr. 5th
- Office hour change

Sun	Mon	Tue	Wed	Thu	Fri
John 1-3	Class 11-12:15		Class 11-12:15 Bert 2-4 Yipeng 4-6		Lauren 11-1

Review

- Frameworks
- The Applet Framework
  - Extend Applet, override init, start, stop, destroy, and paint
- The Collections Framework
  - Collections interface implements Iterable
  - Subinterfaces List and Set
  - AbstractCollection (AbstractList, AbstractSet)

Today's Plan

- (resolve Font drawing confusion)
- Horstmann's graph editor framework
- Prototype pattern
- Example usage of the framework

```
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(); }
```

```
                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;
   private String message;
   private Font font;
   private Rectangle2D bounds;
}
```

getStringBounds vs drawString

- java.awt.Font

```
public Rectangle2D getStringBounds(String str,
                                   FontRenderContext frc)
```

  Returns the logical bounds of the specified String in the specified FontRenderContext. The logical bounds contains the origin, ascent, advance, and height, which includes the leading. The logical bounds does not always enclose all the text. For example, in some languages and in some fonts, accent marks can be positioned above the ascent or below the descent...

- java.awt.Graphics:

```
public abstract void drawString(String str,
                                int x, int y)
```

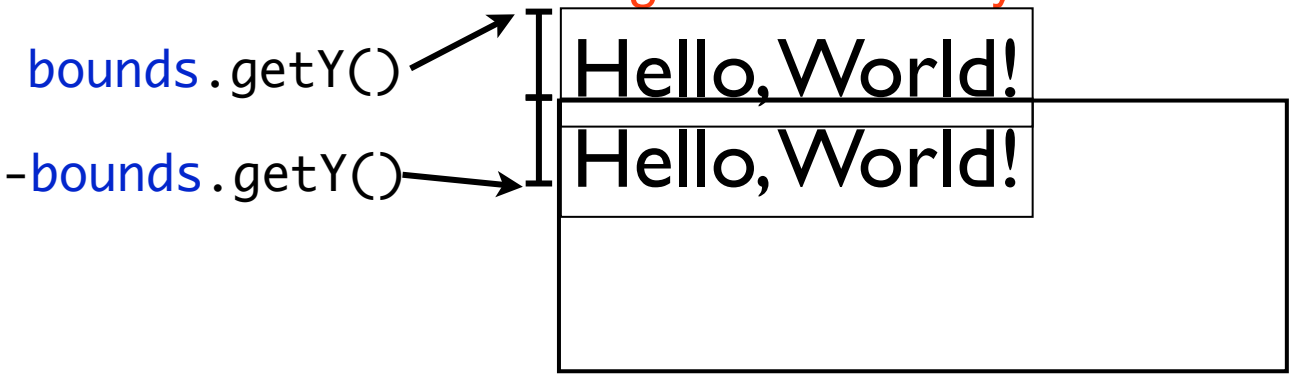
  Draws the text given by the specified string, using this graphics context's current font and color. The baseline of the leftmost character is at position (x, y) in this graphics context's coordinate system.

getStringBounds vs drawString

- java.awt.Font

```
public Rectangle2D getStringBounds(String str,
                                   FontRenderContext frc)
```

Returns the bounding box of text if you call drawString(0, 0) using frc



- java.awt.Graphics:

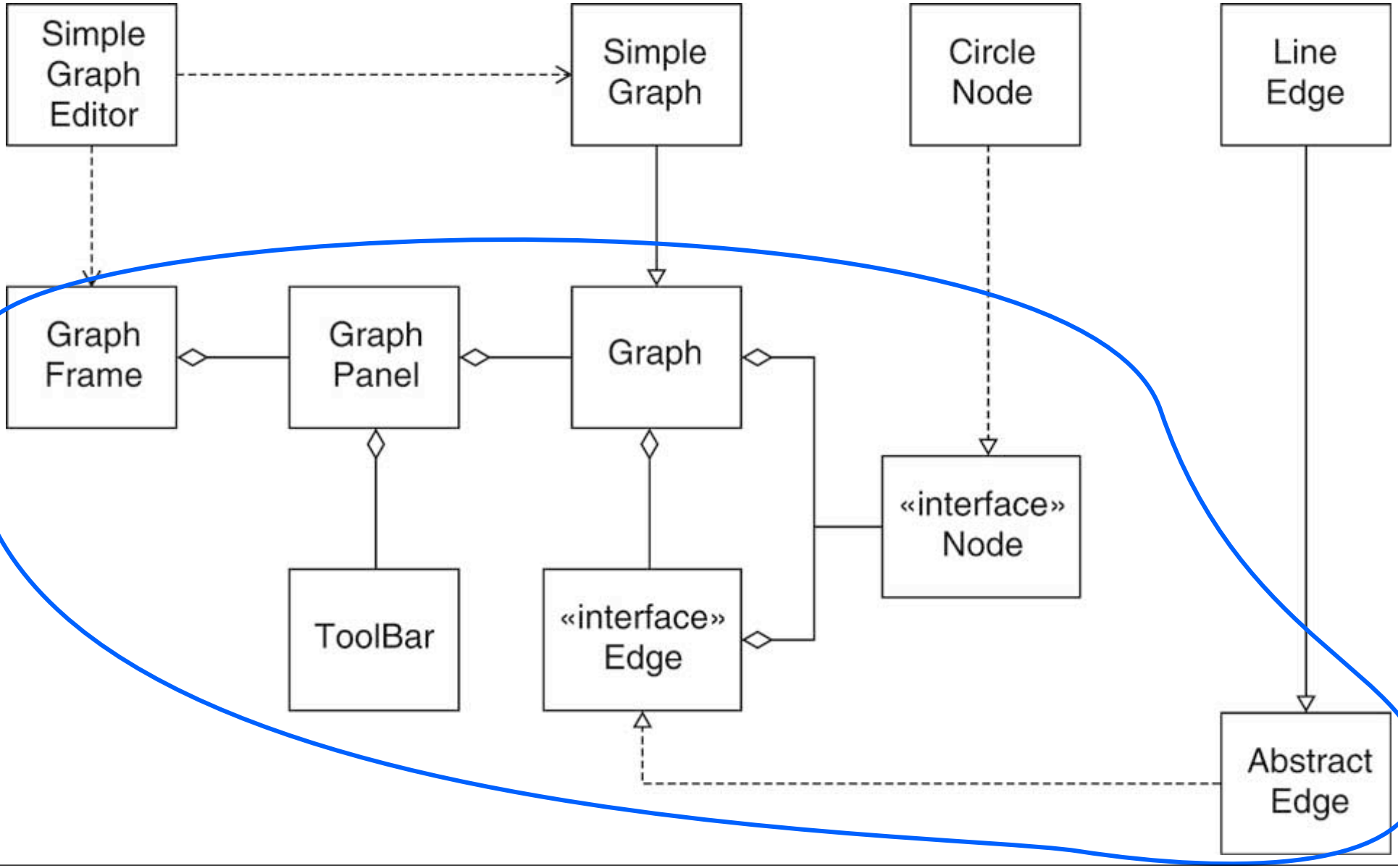
```
public abstract void drawString(String str,
                                int x, int y)
```

Draws the text given by the specified string, using this graphics context's current font and color. The baseline of the leftmost character is at position (x, y) in this graphics context's coordinate system.

Graphs

- Collections of *nodes* connected by *edges*
  - edges may be directed or undirected
- Useful for modeling networks, relationships, states and transitions
  - Nodes can represent people, computers, routers, electrical components, etc
  - Edges can represent friendship, network connectivity, circuits, etc
- We'll look at an application framework for building graph editing programs

Graph Editor Framework



Responsibilities of the Framework vs. Application

- Each application should be responsible for node/edge drawing and *hit testing*
  - Since nodes and edges can represent varying things
- The application should also provide the set of allowed node and edge types

Prototype Pattern

Context

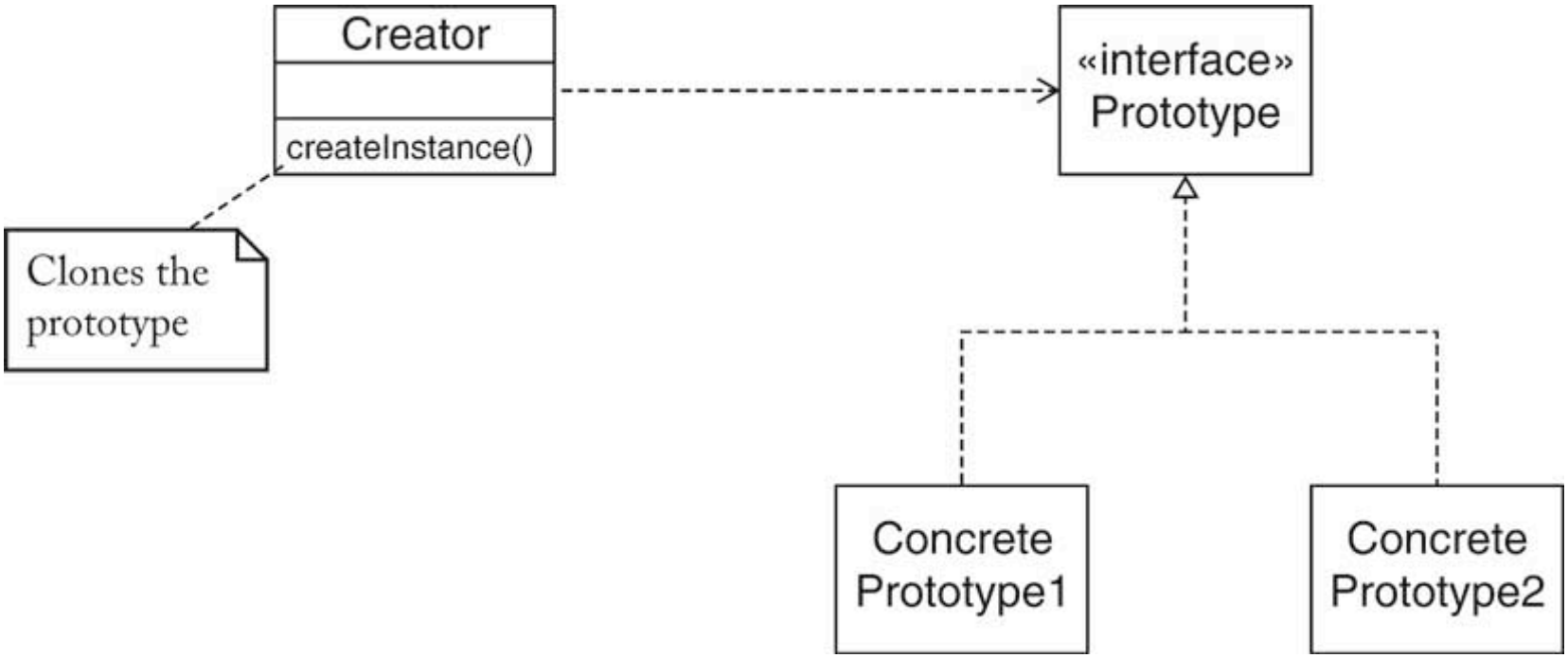
- A system needs to create several kinds of objects whose classes are not known when the system is built
- You don't want to require a separate class for each kind of object
- You want to avoid a separate hierarchy of classes whose responsibility it is to create the objects

---

Solution

- Define a prototype interface common to all created objects
- Supply a prototype object for each kind of object that the system creates
- Clone the prototype object whenever a new object of the given kind is required

Prototype Pattern



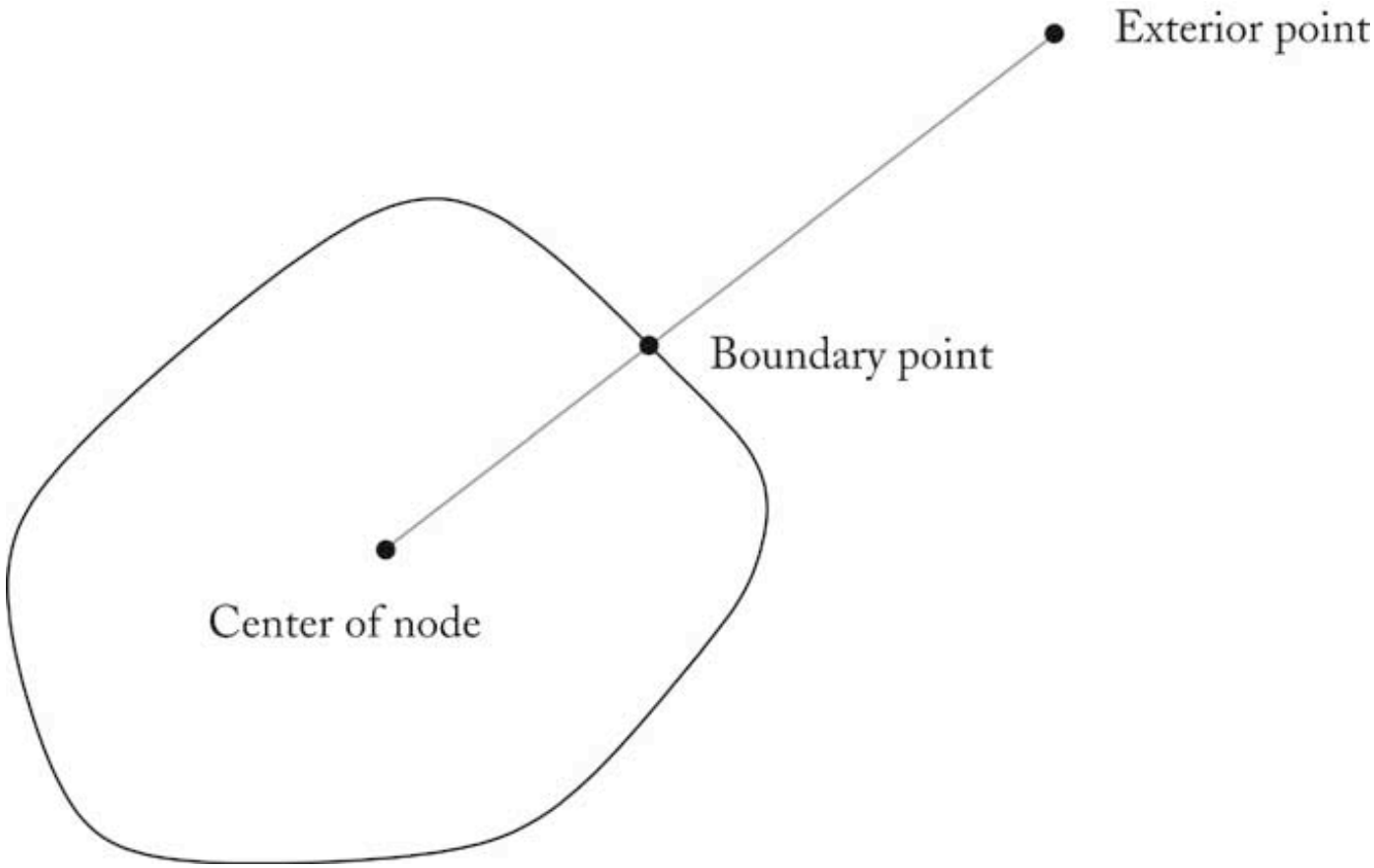
Framework Javadoc

- The framework Horstmann writes is quite extensive already, despite missing some desired features
- We'll examine the classes necessary to write an instance application
  - Node, Edge, Graph

interface Node extends Serializable, Cloneable

Method Summary	
java.lang.Object	**clone**()
boolean	**contains**(java.awt.geom.Point2D aPoint) Tests whether the node contains a point.
void	**draw**(java.awt.Graphics2D g2) Draw the node.
java.awt.geom.Rectangle2D	**getBounds**() Get the bounding rectangle of the shape of this node
java.awt.geom.Point2D	**getConnectionPoint**(java.awt.geom.Point2D aPoint) Get the best connection point to connect this node with another node.
void	**translate**(double dx, double dy) Translates the node by a given amount.

getConnectionPoints



interface Edge extends Serializable, Cloneable

Method Summary	
java.lang.Object	**clone**()
void	**connect**(Node aStart, Node anEnd) Connects this edge to two nodes.
boolean	**contains**(java.awt.geom.Point2D aPoint) Tests whether the edge contains a point.
void	**draw**(java.awt.Graphics2D g2) Draw the edge.
java.awt.geom.Rectangle2D	**getBounds**(java.awt.Graphics2D g2) Gets the smallest rectangle that bounds this edge.
java.awt.geom.Line2D	**getConnectionPoints**() Gets the points at which this edge is connected to its nodes.
Node	**getEnd**() Gets the ending node.
Node	**getStart**() Gets the starting node.

abstract class graph implements Serializable

Constructor Summary

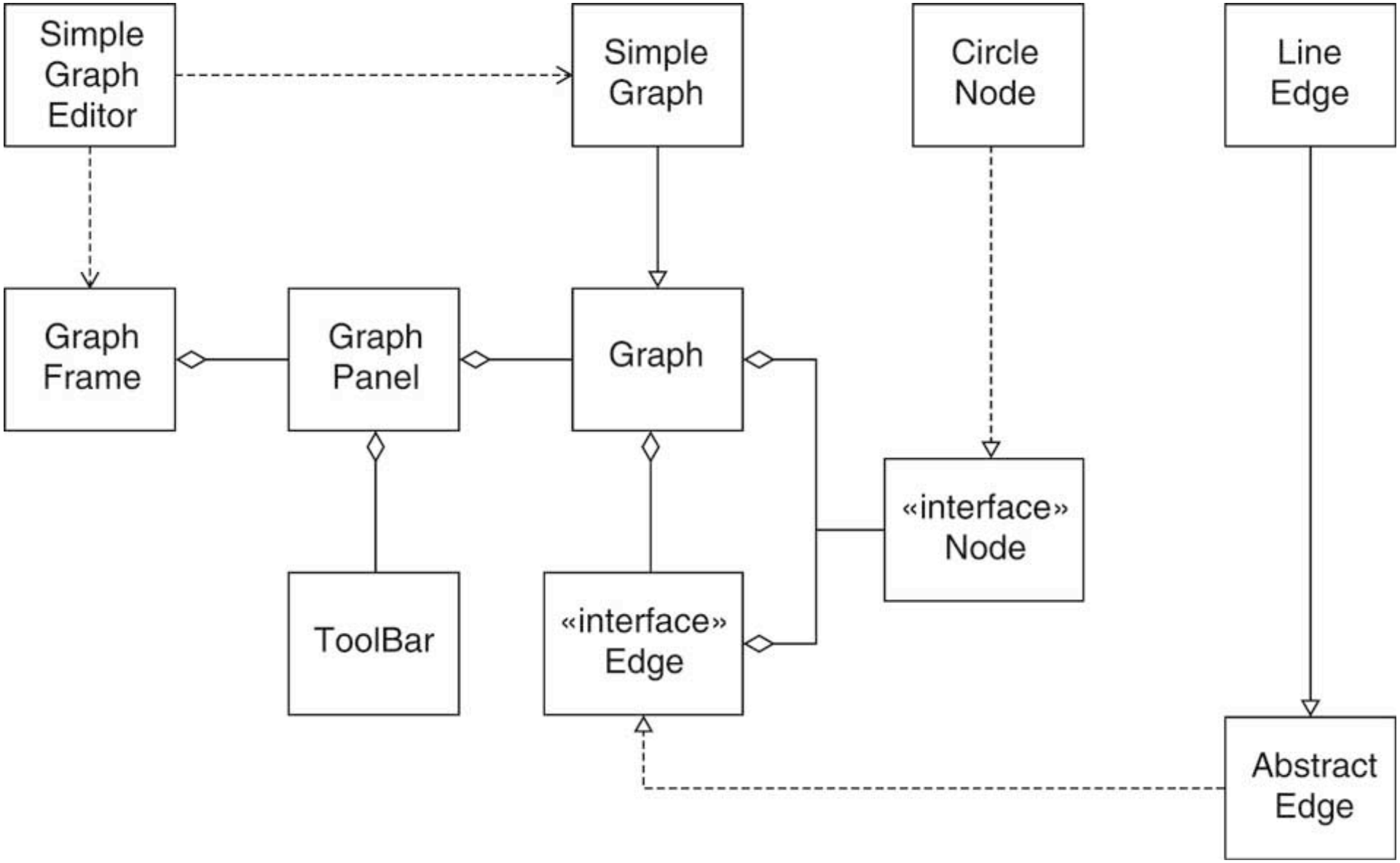
Graph() Constructs a graph with no nodes or edges.	

Method Summary

boolean	**add**(Node n, java.awt.geom.Point2D p) Adds a node to the graph so that the top left corner of the bounding rectangle is at the given point.
boolean	**connect**(Edge e, java.awt.geom.Point2D p1, java.awt.geom.Point2D p2) Adds an edge to the graph that joins the nodes containing the given points.
void	**draw**(java.awt.Graphics2D g2) Draws the graph
Edge	**findEdge**(java.awt.geom.Point2D p) Finds an edge containing the given point.
Node	**findNode**(java.awt.geom.Point2D p)

	rectangle is at the given point.
boolean	**connect**(Edge e, java.awt.geom.Point2D p1, java.awt.geom.Point2D p2) Adds an edge to the graph that joins the nodes containing the given points.
void	**draw**(java.awt.Graphics2D g2) Draws the graph
Edge	**findEdge**(java.awt.geom.Point2D p) Finds an edge containing the given point.
Node	**findNode**(java.awt.geom.Point2D p) Finds a node containing the given point.
java.awt.geom.Rectangle2D	**getBounds**(java.awt.Graphics2D g2) Gets the smallest rectangle enclosing the graph
abstract Edge[]	**getEdgePrototypes**() Gets the edge types of a particular graph type.
java.util.List<Edge>	**getEdges**() Gets the edges of this graph.
abstract Node[]	**getNodePrototypes**() Gets the node types of a particular graph type.
java.util.List<Node>	**getNodes**() Gets the nodes of this graph.
void	**removeEdge**(Edge e) Removes an edge from the graph.
void	**removeNode**(Node n) Removes a node and all edges that start or end with that node

Simple Graph Editor



SimpleGraphEditor

```
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.setVisible(true);
   }
}
```

SimpleGraph

```
import java.awt.*;
import java.util.*;
/**
   A simple graph with round nodes and
straight edges.
*/
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;
   }
}
```

LineEdge

```
/**
   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;
   }
}
```

SimpleGraph

```
import java.awt.*;
import java.awt.geom.*;

/**
   A circular node that is filled with a color.
*/
public class CircleNode implements Node
{
   /**
      Construct a circle node with a given size and color.
      @param aColor the fill color
   */
   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)
```

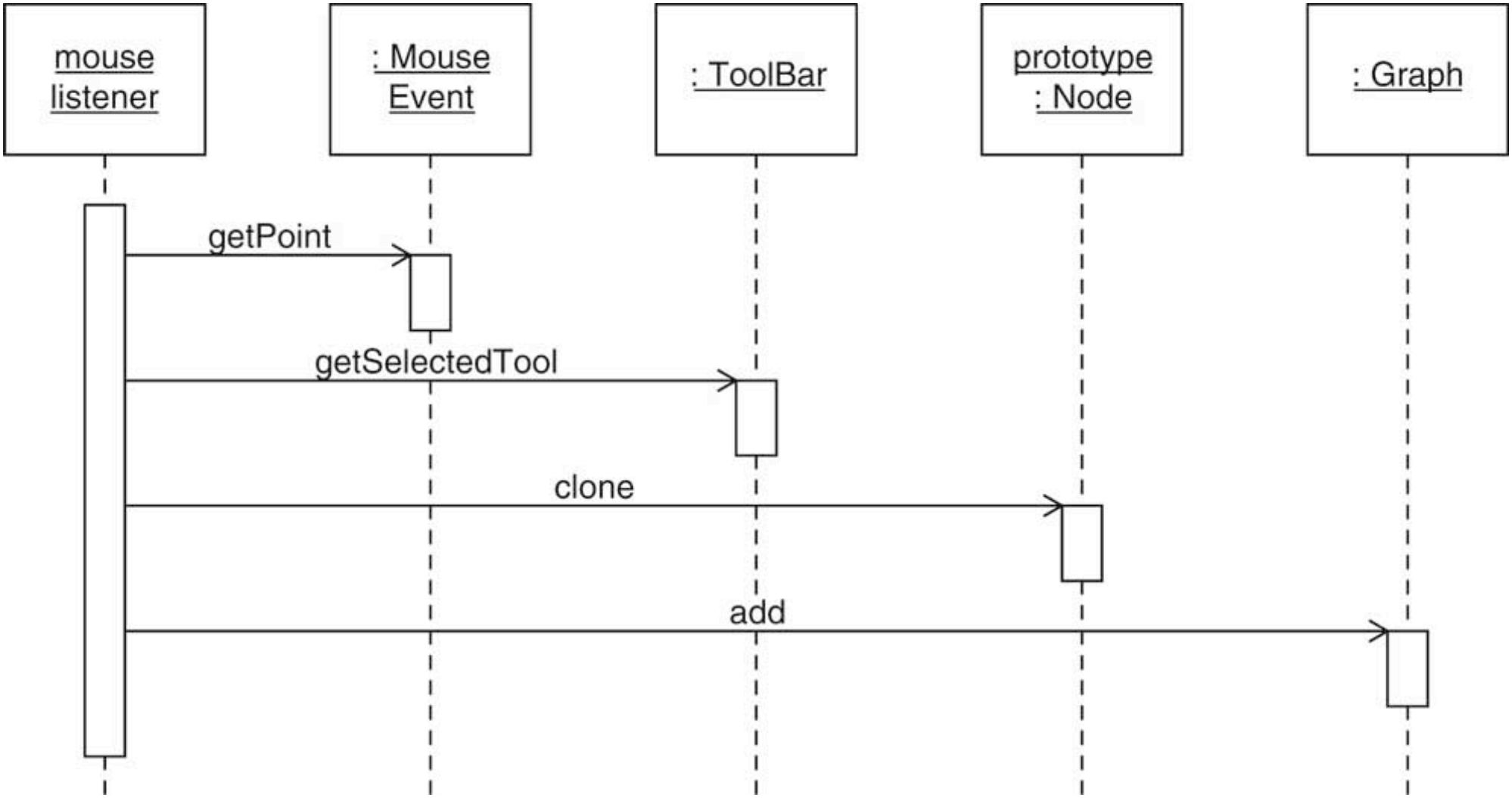
```
public void translate(double dx, double dy)
{
   x += dx;
   y += dy;
}

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;
```

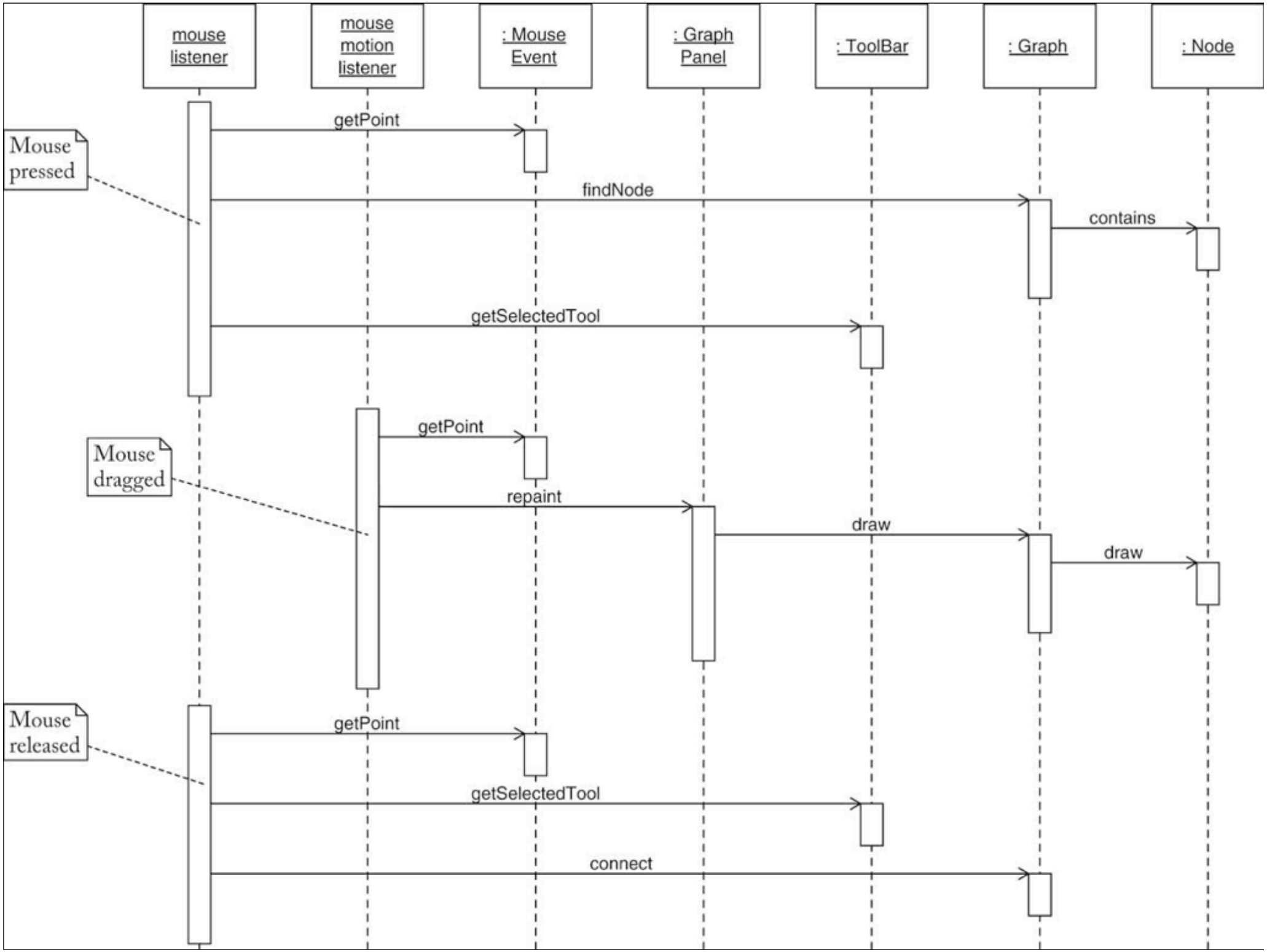
```
      return new Rectangle2D.Double(
            x, y, size, size);
   }

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

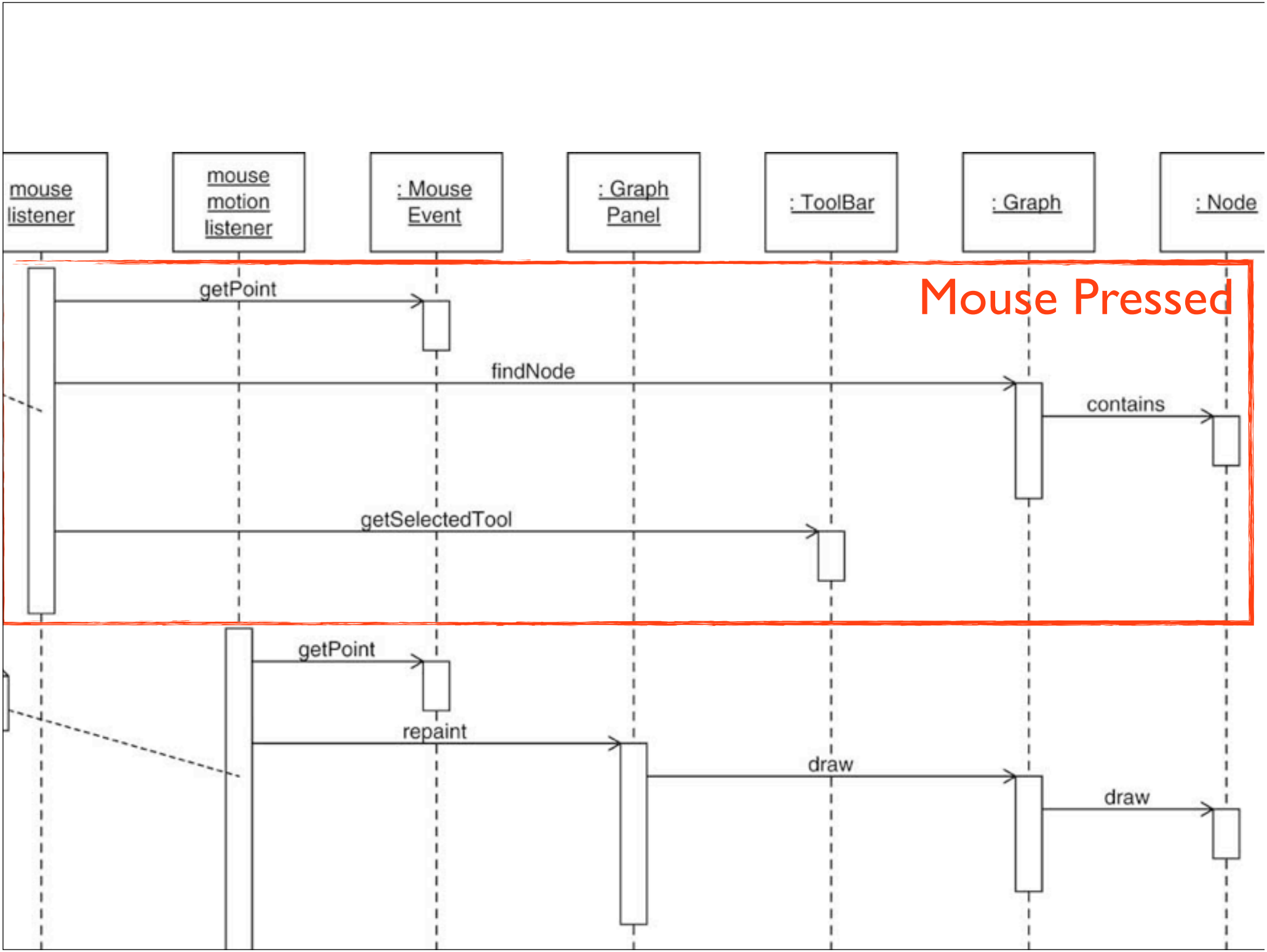
Framework Generics

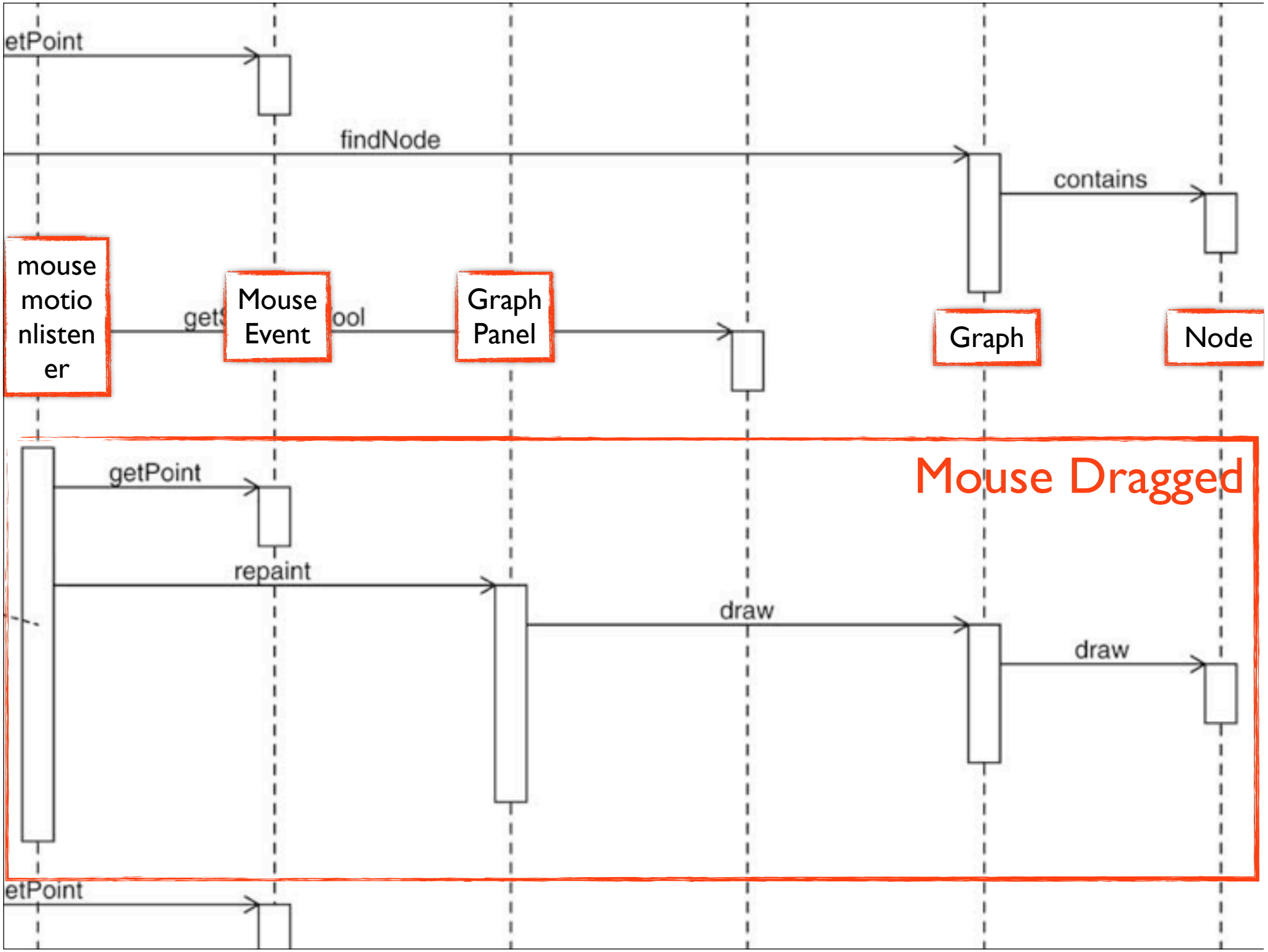
- The framework code does not need to know the concrete node, edge or graph types
- Important operations can use the template-method pattern and let the concrete classes handle the specifics

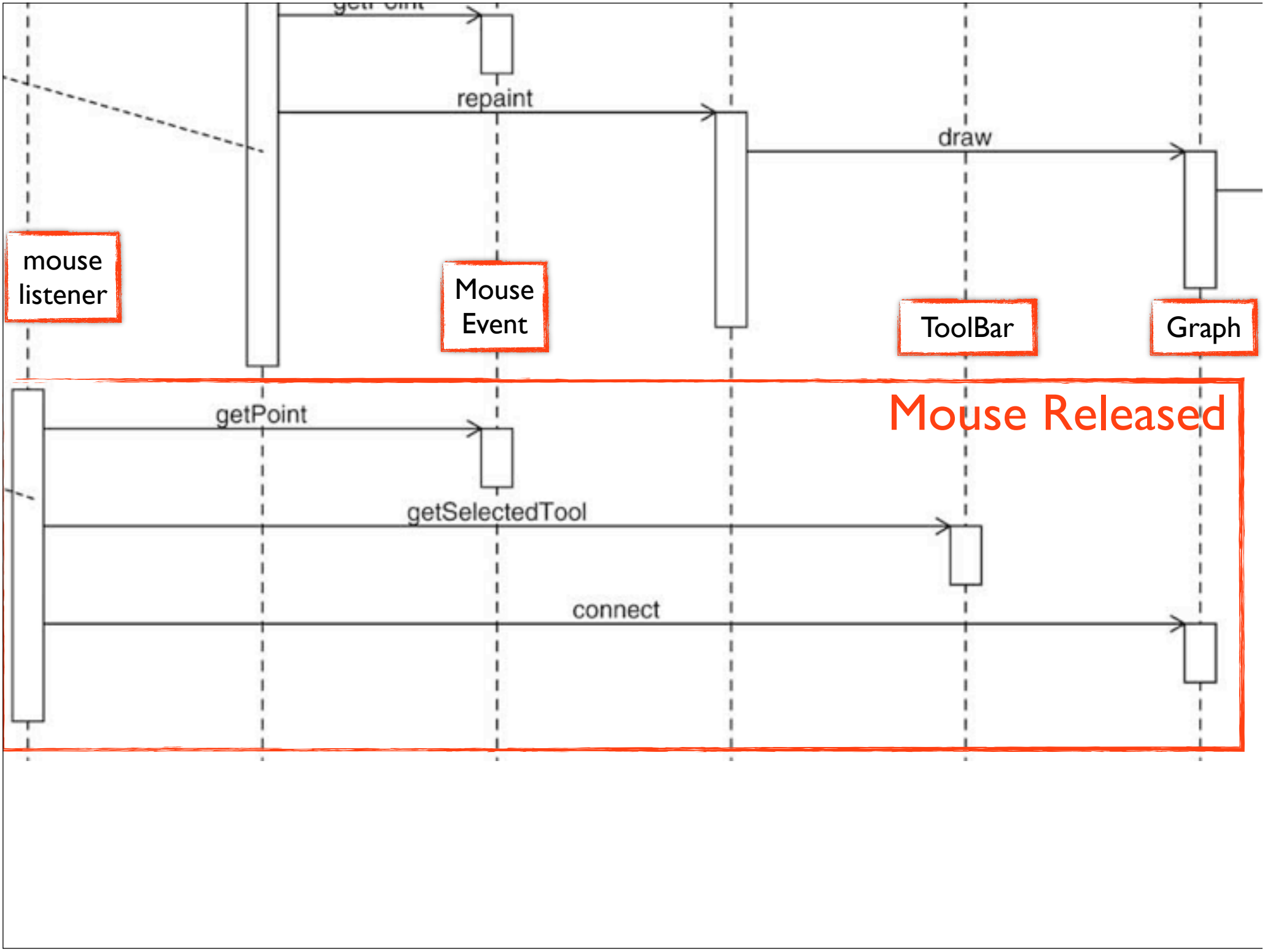
Adding a Node

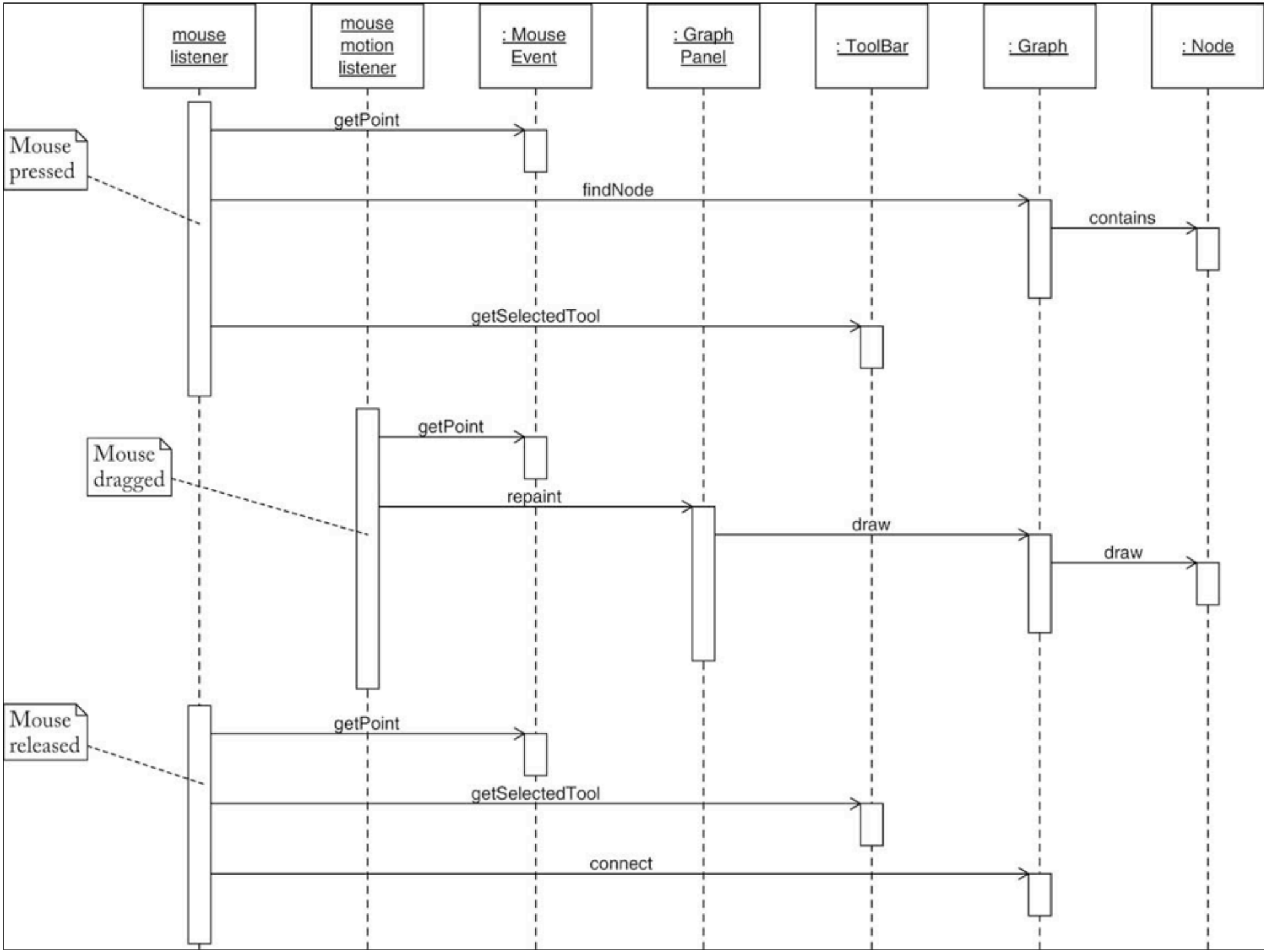



mouse listener
mouse motion listener
: Mouse Event
: Graph Panel
: ToolBar
: Graph
: Node
Mouse pressed
getPoint
findNode
contains
getSelectedTool
Mouse dragged
getPoint
repaint
draw
draw
Mouse released
getPoint
getSelectedTool
connect


mouse listener
mouse motion listener
: Mouse Event
: Graph Panel
: ToolBar
: Graph
: Node
Mouse Pressed
getPoint
findNode
contains
getSelectedTool
getPoint
repaint
draw
draw


etPoint
findNode
contains
mouse motio nlisten er
Mouse Event
Graph Panel
Graph
Node
getPoint
repaint
draw
draw
Mouse Dragged
etPoint


repaint
draw
mouse listener
Mouse Event
ToolBar
Graph
Mouse Released
getPoint
getSelectedTool
connect


mouse listener
mouse motion listener
: Mouse Event
: Graph Panel
: ToolBar
: Graph
: Node
Mouse pressed
getPoint
findNode
contains
getSelectedTool
Mouse dragged
getPoint
repaint
draw
draw
Mouse released
getPoint
getSelectedTool
connect

Reading

- Horstmann Ch. 8.4