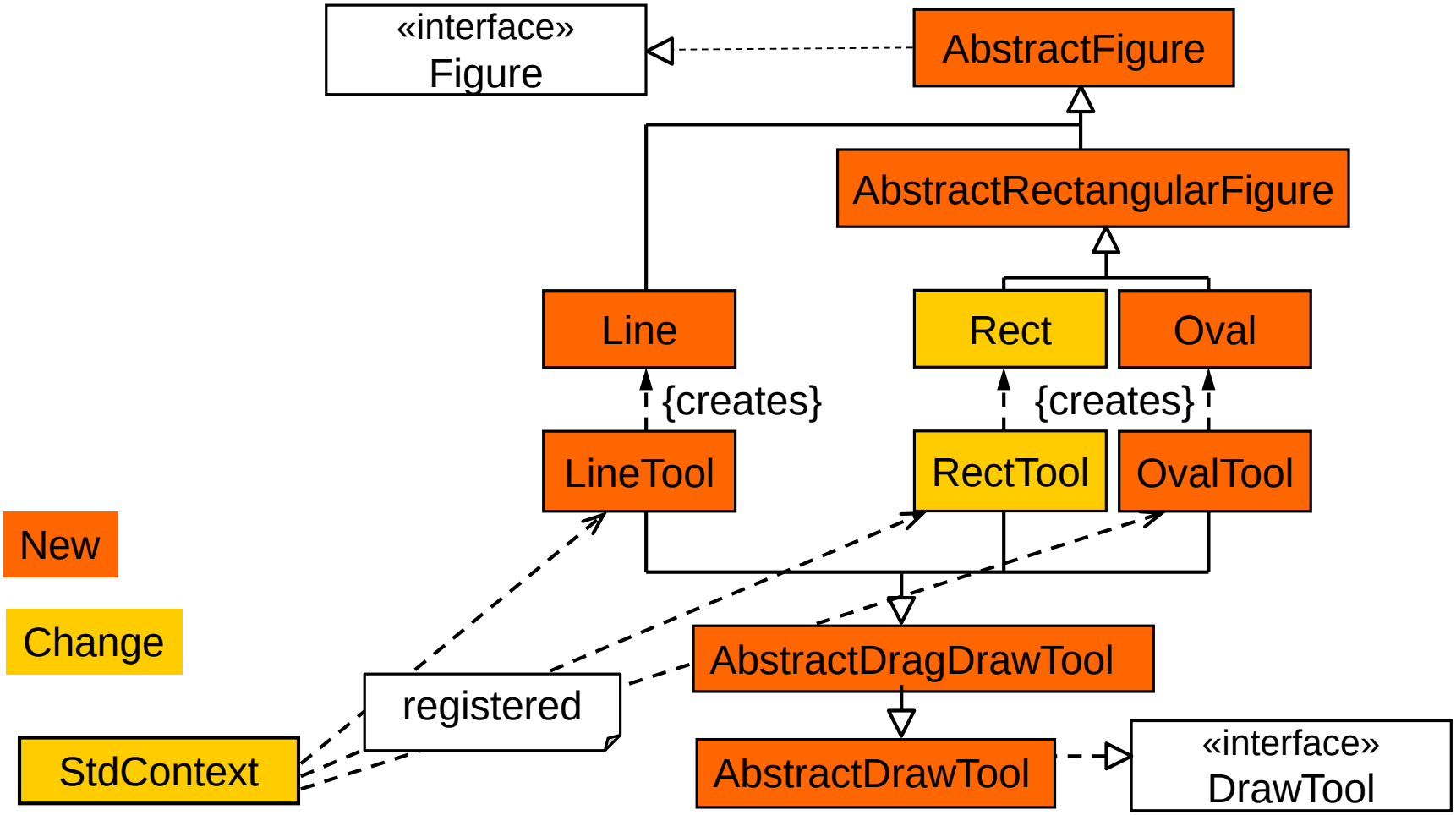
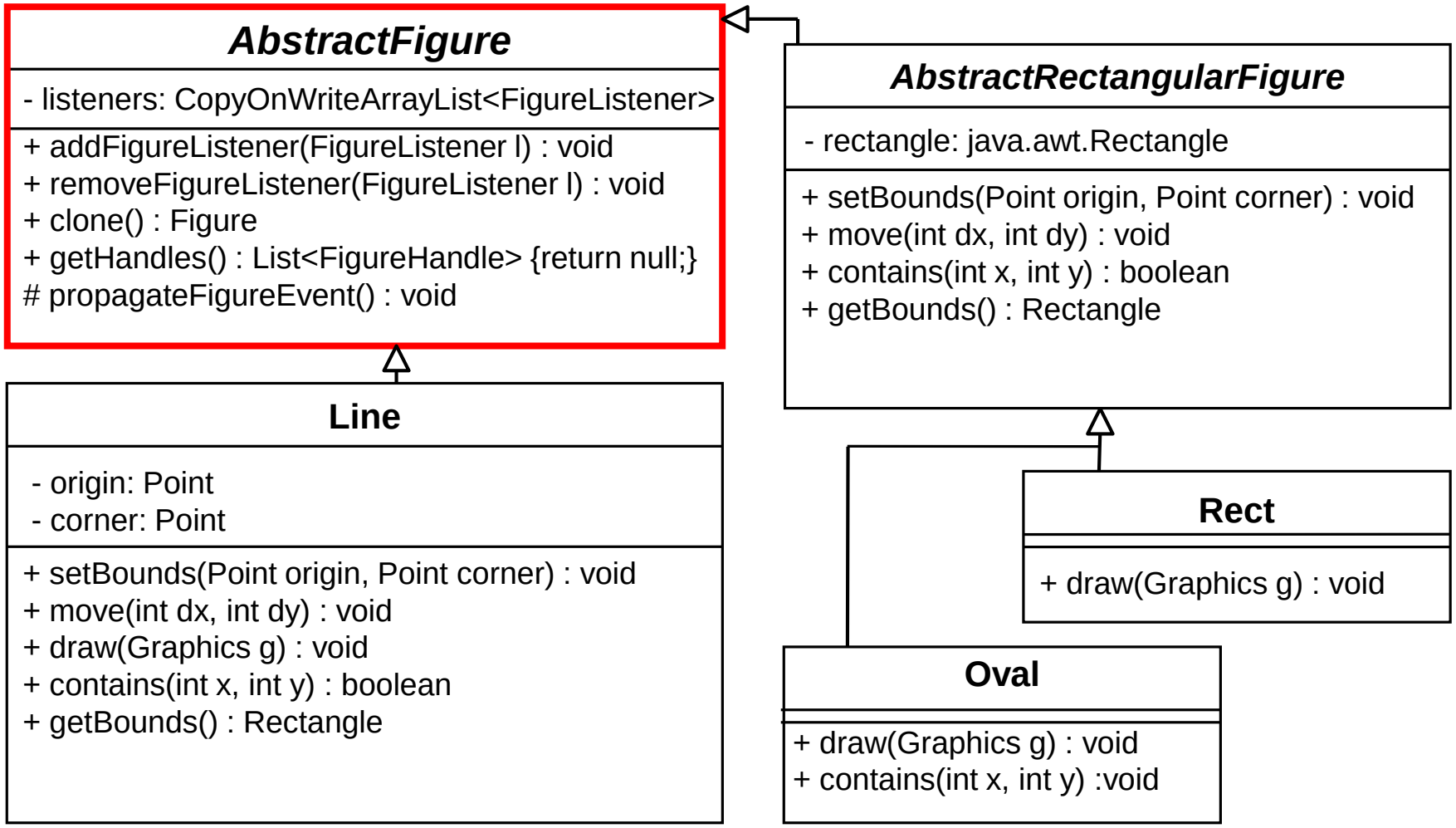
Assignment 3: Figures



Figures



AbstractFigure (1/2)

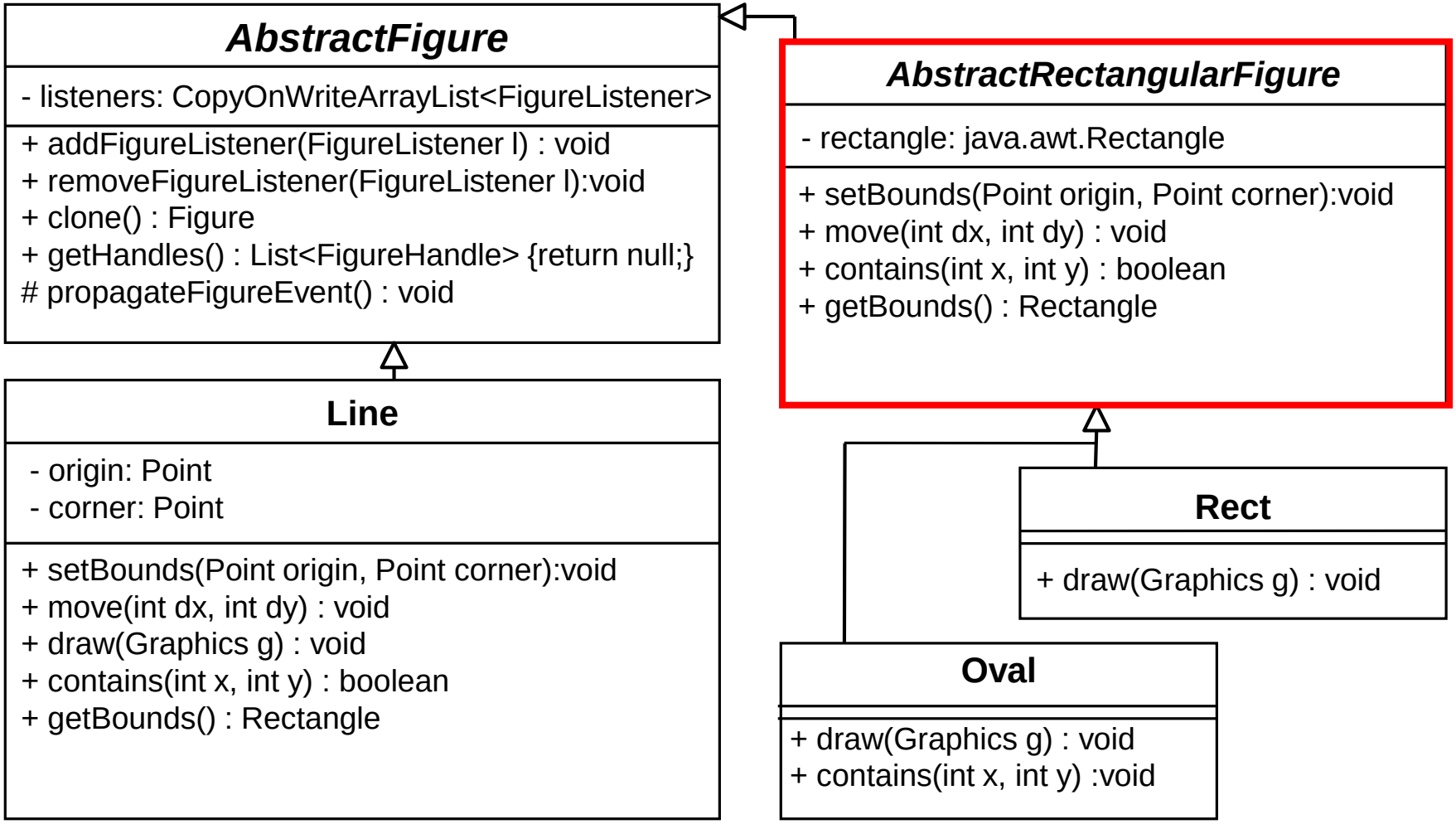
```
public abstract class AbstractFigure implements Figure {

   private final List<FigureListener> listeners
                                    = new CopyOnWriteArrayList <>();

   @Override
   public final void addFigureListener(FigureListener listener) {
      if (listener != null && !listeners.contains(listener)) {
         listeners.add(listener);
      }
   }

   @Override
   public final void removeFigureListener(FigureListener listener) {
      listeners.remove(listener);
   }

   …
```

AbstractFigure (2/2)

```
    protected final void propagateFigureEvent() {
        FigureEvent event = new FigureEvent(this);
        for (FigureListener l : listeners) { l.figureChanged(event); }
    }

    @Override
    public Figure clone() {
        return null;
    }

    @Override
    public List<FigureHandle> getHandles() {
        return null;
    }

}
```

No need to iterate over a copy as a `CopyOnWriteArrayList` is used.

Figures



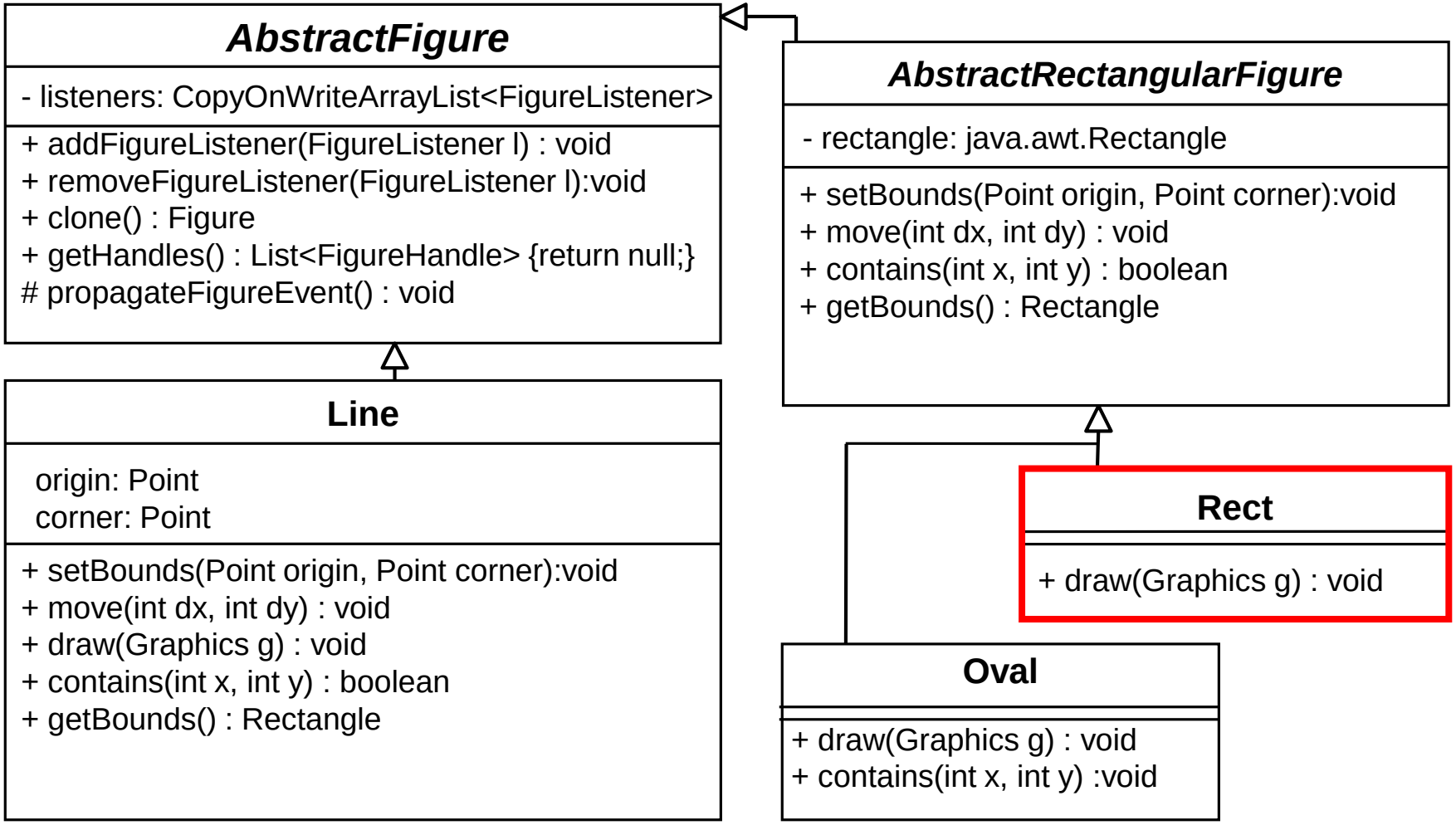
AbstractRectangularFigure (1/2)

```
public abstract class AbstractRectangularFigure extends AbstractFigure {
   private final Rectangle rectangle;

   protected AbstractRectangularFigure(Point origin) {
      rectangle = new Rectangle(origin);
   }

   @Override
   public void setBounds(Point origin, Point corner) {
      Rectangle original = new Rectangle(rectangle);
      rectangle.setFrameFromDiagonal(origin, corner);
      if (!original.equals(rectangle)) { // notification only if
         propagateFigureEvent();         // there is a change
      }
   }

   …
```

AbstractRectangularFigure (2/2)

```
   @Override
   public void move(int dx, int dy) {
      if (dx != 0 || dy != 0) { // notification only if changed
         rectangle.translate(dx, dy);
         propagateFigureEvent();
      }
   }

   @Override
   public boolean contains(int x, int y) {
      return rectangle.contains(x, y);
   }

   @Override
   public Rectangle getBounds() {
      return new Rectangle(rectangle);
   }
}
```

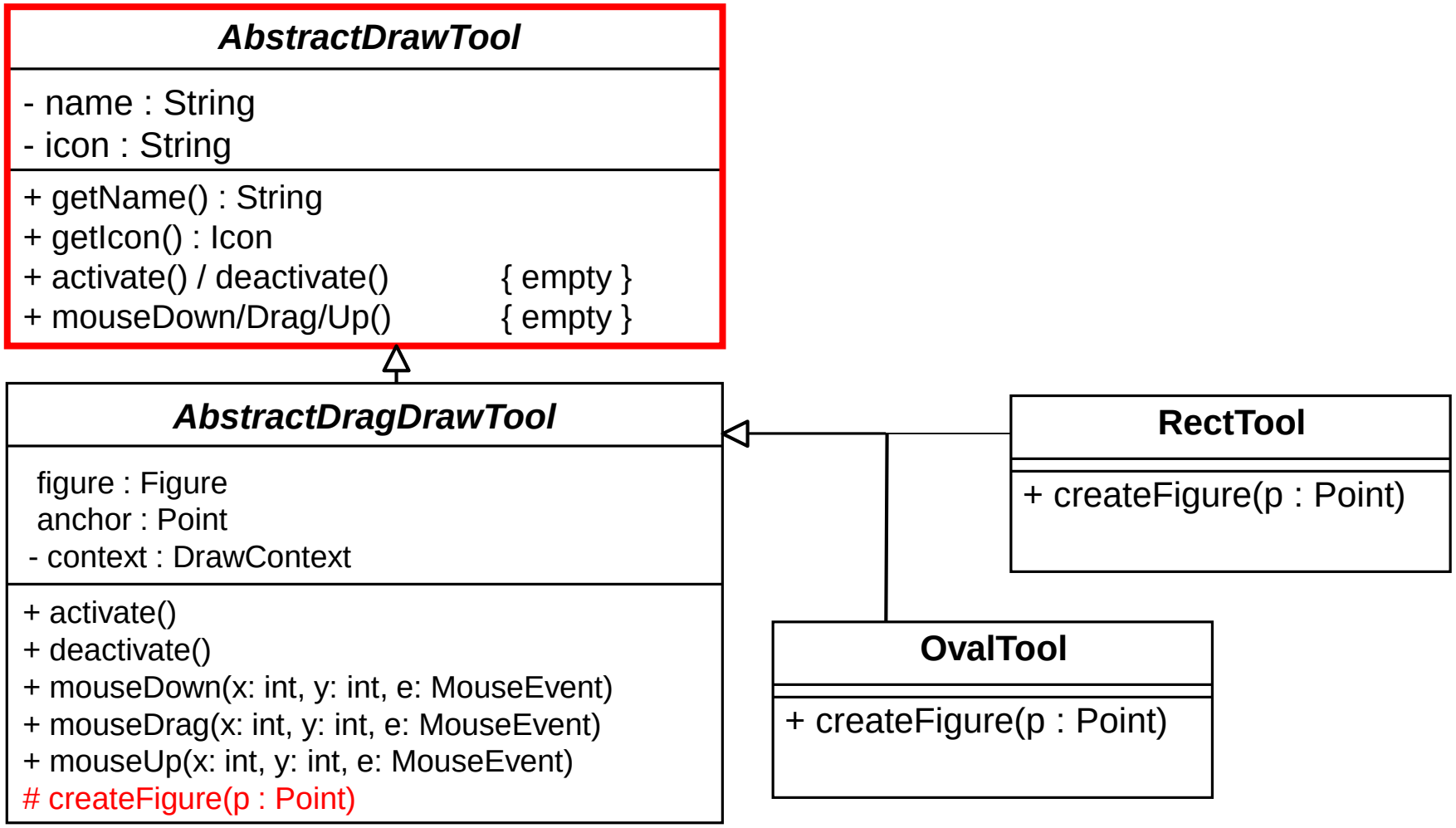
Figures



Rect

```
public class Rect extends AbstractRectangularFigure {

   public Rect(Point p) { super(p); }

   @Override
   public void draw(Graphics g){
      Rectangle r = getBounds();
      g.setColor(Color.white);
      g.fillRect(r.x, r.y, r.width, r.height);
      g.setColor(Color.black);
      g.drawRect(r.x, r.y, r.width, r.height);
   }

}
```

Tools



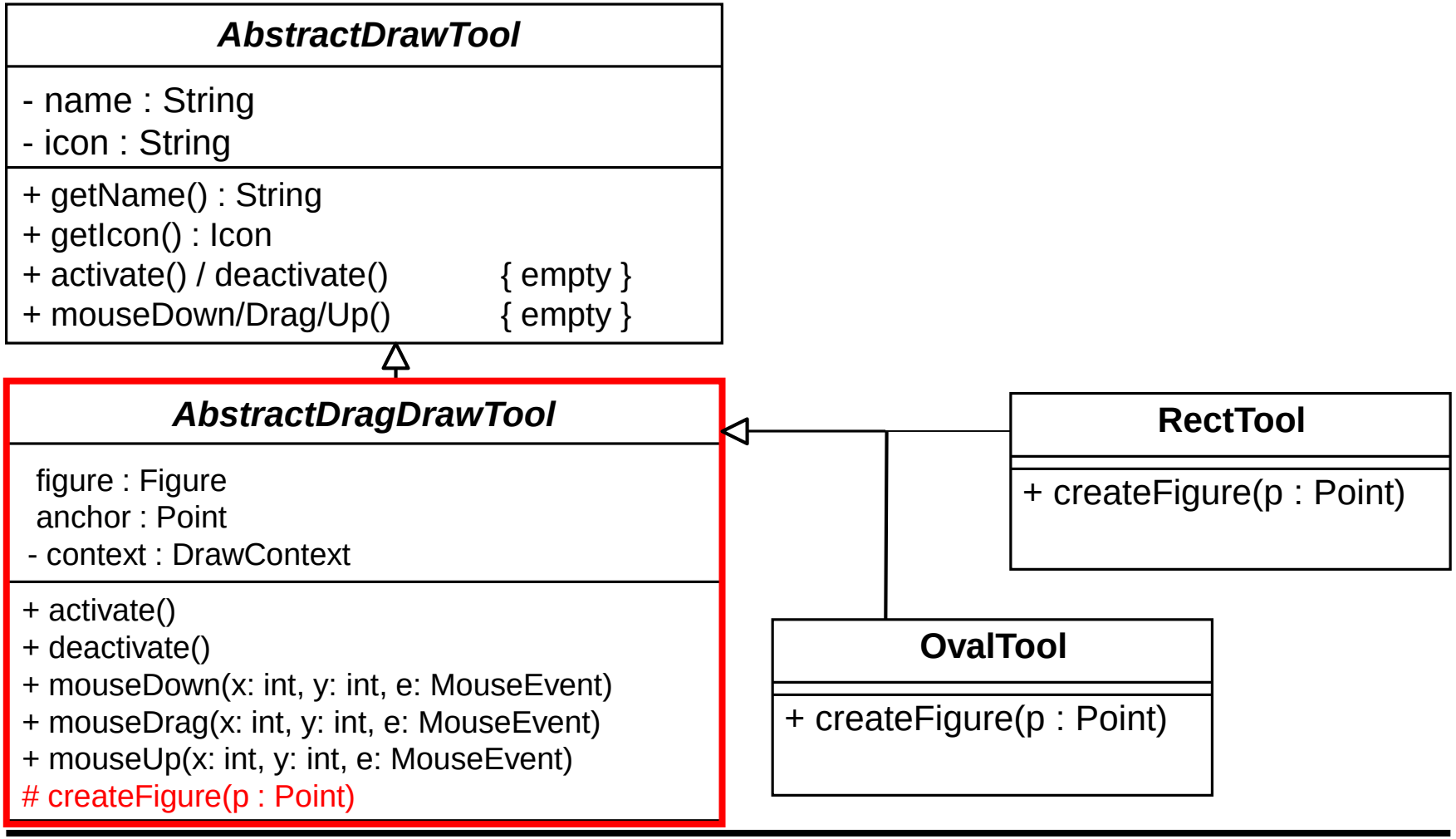
AbstractDrawTool (1/2)

```
public abstract class AbstractDrawTool implements DrawTool {
   private static final String IMAGES = "/images/";
   private final String name;
   private final String icon;

   protected AbstractDrawTool(String name, String icon) {
      this.name = name; this.icon = icon;
   }

   @Override
   public final String getName() { return name; }

   @Override
   public final Icon getIcon() {
      if (icon != null) {
        return new ImageIcon(getClass().getResource(IMAGES+icon));
      } else { return null; }
   }
```

AbstractDrawTool (2/2)

```
   @Override
   public Cursor getCursor() {
      return Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR);
   }

   @Override
   public void activate() { }

   @Override
   public void deactivate() { }

   @Override
   public void mouseDown(int x, int y, MouseEvent e) { }

   @Override
   public void mouseDrag(int x, int y, MouseEvent e) { }

   @Override
   public void mouseUp(int x, int y, MouseEvent e) { }
}
```

Tools



AbstractDragDrawTool (1/3)

```
public abstract class AbstractDragDrawTool extends AbstractDrawTool {
   private final DrawContext context;
   private Point anchor;
   private Figure figure;

   protected AbstractDragDrawTool(DrawContext context,
                                  String name, String icon) {
      super(name, icon); this.context = context;
   }

   @Override
   public void activate() {
      context.showStatusText(getName() + " Mode");
   }

   @Override
   public void deactivate() {
      context.showStatusText("");
   }
```

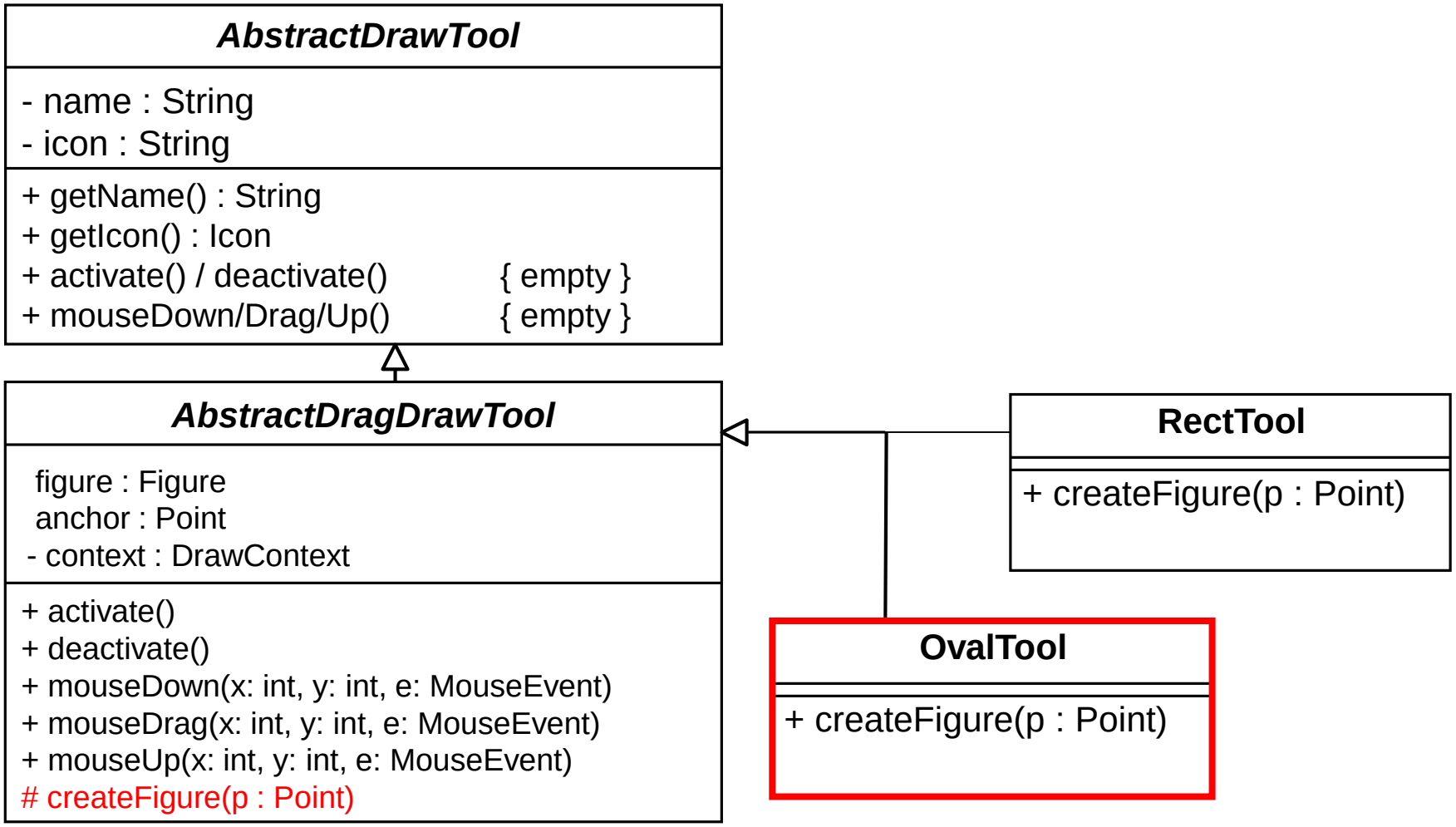
AbstractDragDrawTool (2/3)

```
protected abstract Figure createFigure(Point p);

@Override
public final void mouseDown(int x, int y, MouseEvent e) {
   if (figure != null) {
      throw new IllegalStateException();
   }
   anchor = new Point(x, y);
   figure = createFigure(anchor);
   context.getModel().addFigure(figure);
}

@Override
public final void mouseDrag(int x, int y, MouseEvent e) {
   figure.setBounds(anchor, new Point(x, y));
}
```

AbstractDragDrawTool (3/3)

```
   @Override
   public final void mouseUp(int x, int y, MouseEvent e) {
      Rectangle r = figure.getBounds();
      if (r.width == 0 && r.height == 0) {
         context.getModel().removeFigure(figure);
      }
      anchor = null;
      figure = null;
   }
}
```

Tools



OvalTool

```
public class OvalTool extends AbstractDragDrawTool {

   public OvalTool(DrawContext context, String name, String icon) {
      super(context, name, icon);
   }

   @Override
   protected Oval createFigure(Point p) {
      return new Oval(p);
   }
}
```