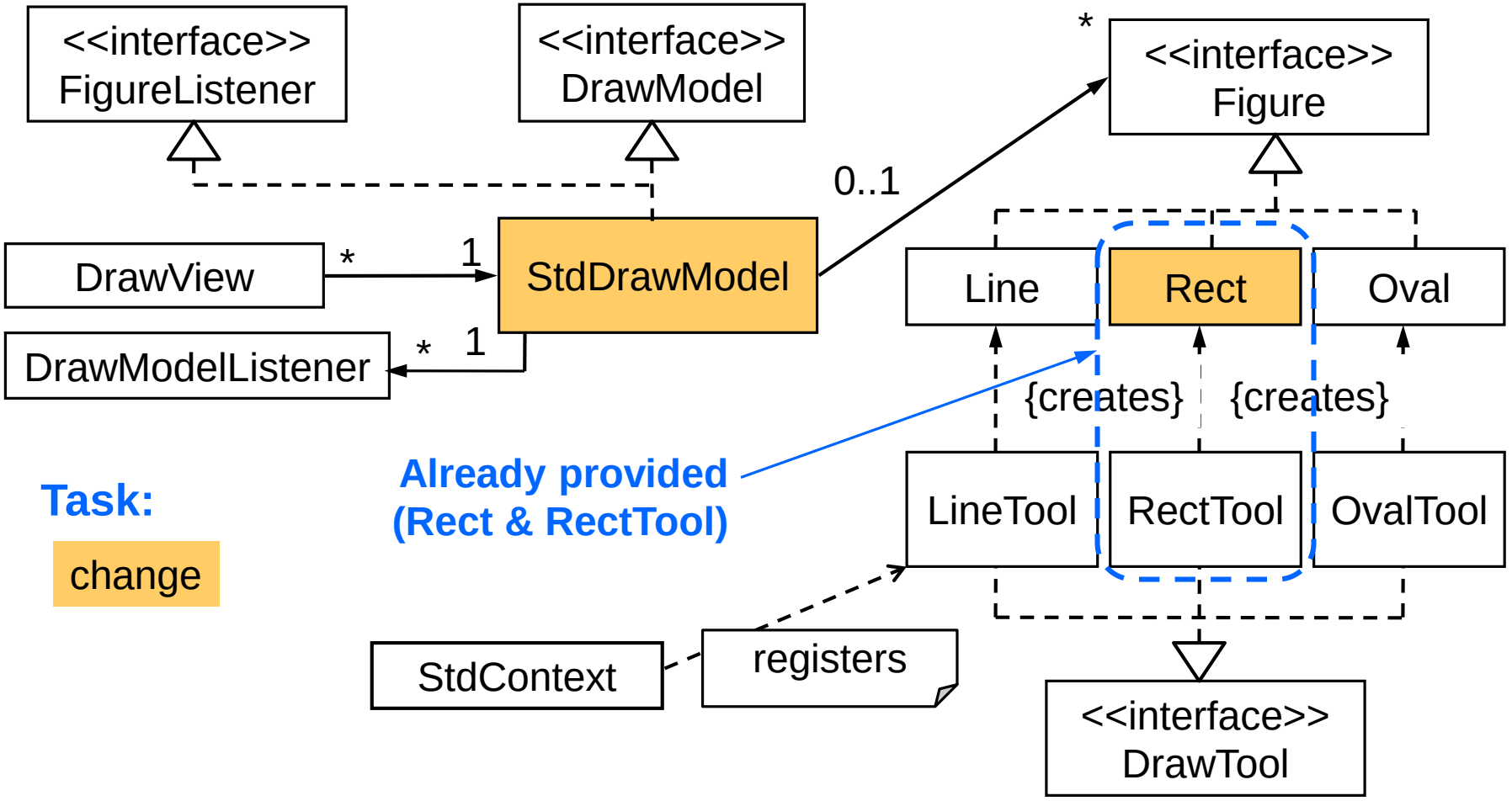
Assignment 2: Observer Pattern



DrawModel (1/4)

```
public class StdDrawModel implements DrawModel, FigureListener {
   /** List of figures contained in the drawing. */
   private final List<Figure> figures = new LinkedList<>();

   /** List of listeners interested in changes of any figure. */
   private final List<DrawModelListener> listeners
                                 = new CopyOnWriteArrayList<>();
   @Override
   public void addModelChangeListener(DrawModelListener l) {
      if (l != null && !listeners.contains(l)) {
         listeners.add(l);
      }
   }

   @Override
   public void removeModelChangeListener(DrawModelListener l) {
      listeners.remove(l);
   }
```

DrawModel (2/4)

```
protected void notifyListeners(Figure f,
                               DrawModelEvent.Type type) {
   DrawModelEvent dme = new DrawModelEvent(this, f, type);
   for(DrawModelListener l : listeners) {
      l.modelChanged(dme);
   }
}

@Override
public void figureChanged(FigureEvent e) {
   notifyListeners(e.getFigure(),
                   DrawModelEvent.Type.FIGURE_CHANGED);
}

@Override
public Iterable<Figure> getFigures() {
   return Collections.unmodifiableList(figures);
}
```

If the listener list were not a *Copy-On-Write List*, it would be necessary to iterate over a copy in order to prevent a CME

DrawModel (2/4)

```
protected void notifyListeners(Figure f,
                               DrawModelEvent.Type type) {
   DrawModelEvent dme = new DrawModelEvent(this, f, type);
   listeners.forEach(l -> l.modelChanged(dme));
}


@Override
public void figureChanged(FigureEvent e) {
   notifyListeners(e.getFigure(),
                   DrawModelEvent.Type.FIGURE_CHANGED);
}

@Override
public Iterable<Figure> getFigures() {
   return Collections.unmodifiableList(figures);
}
```

If the listener list were not a *Copy-On-Write List*, it would be necessary to iterate over a copy in order to prevent a CME

DrawModel (3/4)

```
@Override
public void addFigure(Figure f) {
   if (f != null && !figures.contains(f)) {
      figures.add(f);
      f.addFigureListener(this);
      notifyListeners(f, DrawModelEvent.Type.FIGURE_ADDED);
   }
}

@Override
public void removeFigure(Figure f) {
   if (figures.remove(f)) {
      f.removeFigureListener(this);
      notifyListeners(f, DrawModelEvent.Type.FIGURE_REMOVED);
   }
}
```

DrawModel (4/4)

```
    @Override
    public void setFigureIndex(Figure f, int index) {
        if(index < 0 || index >= figures.size()) {
            throw new IndexOutOfBoundsException();
        }
        int pos = figures.indexOf(f);
        if (pos < 0) {
            throw new IllegalArgumentException(
                    "Figure f not contained in model");
        }
        if (pos != index) {
            figures.remove(f);
            figures.add(index, f);
            notifyListeners(f, DrawModelEvent.Type.DRAWING_CHANGED);
        }
    }
    ...
}
```

Rect (1/2)

```
public class Rect implements Figure {
   private final List<FigureListener> listeners =
                                    new CopyOnWriteArrayList<>();
   @Override
   public void addFigureListener(FigureListener listener) {
      if(listener != null && !listeners.contains(listener)) {
         listeners.add(listener);
      }
   }

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

   protected void propagateFigureEvent(FigureEvent evt) {
      listeners.forEach(l -> l.figureChanged(evt));
   }
```

Rect (2/2)

```
private java.awt.Rectangle rect;

@Override
public void move(int dx, int dy) {
   if(dx != 0 || dy != 0) {
      rect.setLocation(rect.x + dx, rect.y + dy);
      propagateFigureEvent(new FigureEvent(this));
   }
}

@Override
public void setBounds(Point origin, Point corner) {
   java.awt.Rectangle original = new java.awt.Rectangle(rect);
   rect.setFrameFromDiagonal(origin, corner);
   if(!original.equals(rect)) {
      propagateFigureEvent(new FigureEvent(this));
   }
}
```

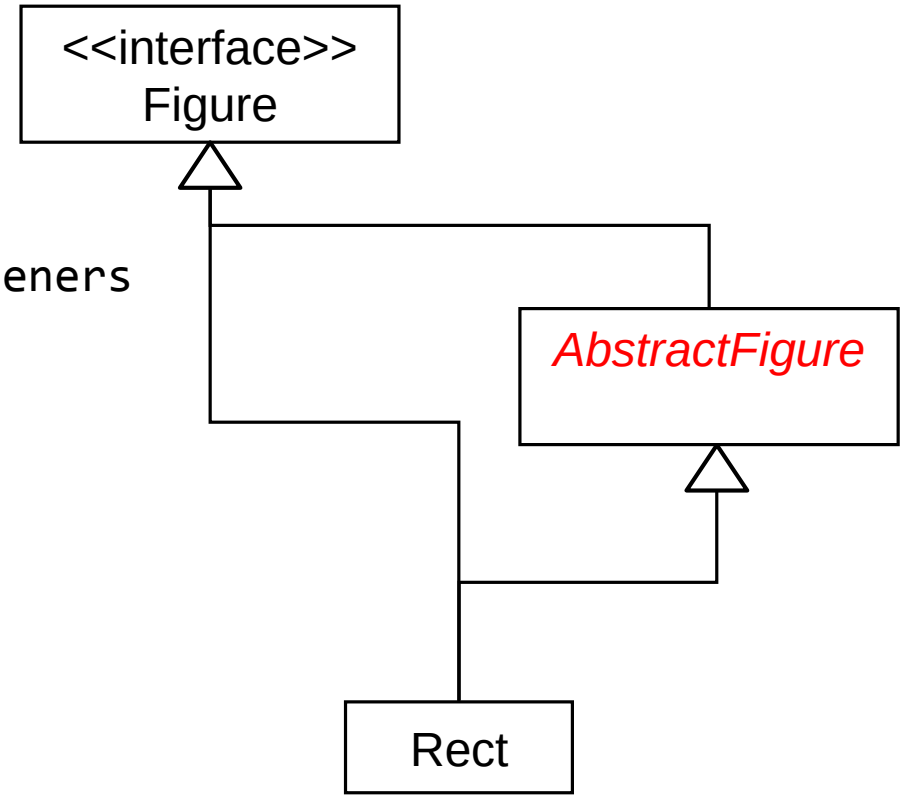
AbstractFigure

Abstract Base Class

Code for the administration of the listeners is identical for all figures:

- `List<FigureListeners> listeners`
- `addFigureListener`
- `removeFigureListener`
- `propagateFigureEvent`

=> Is best defined in an abstract base class



Assignments: Overview

Implementation of the DrawModel

U2

Implementation of additional Figures

- Line
- Oval

U3

Implementation of the Figure Handles

- Allows to modify the figures

U4

Assignment 4: Implementation of Handles

Handles

- Can be selected with the mouse
- With dragging of the handles the figures are changed

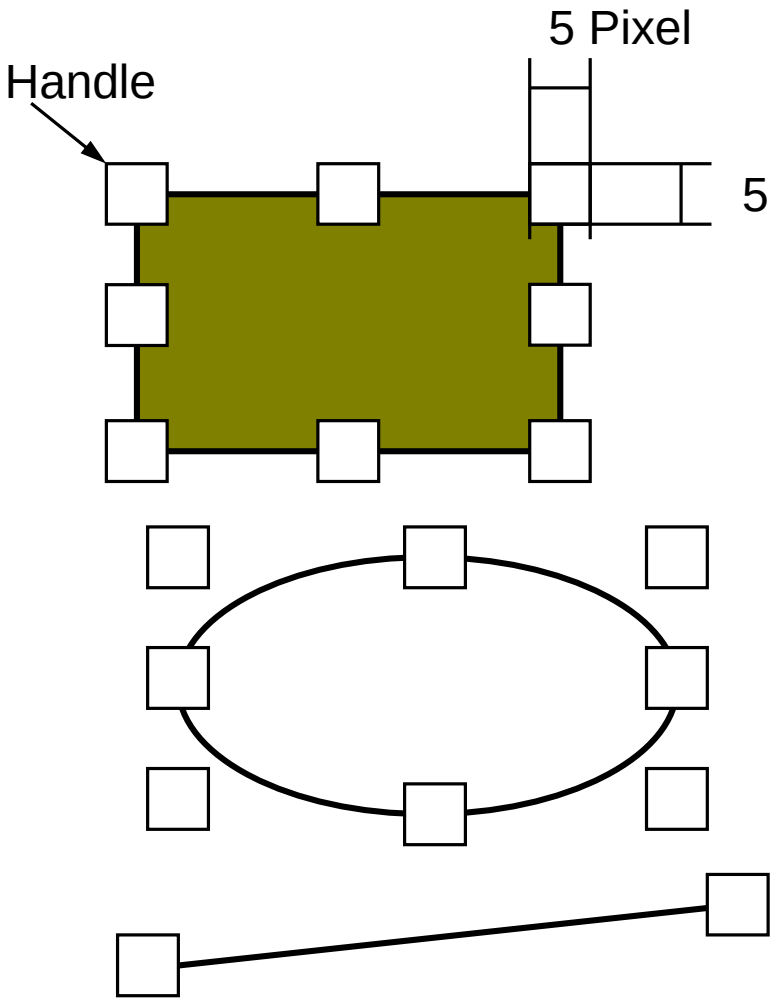
- Given implementation:

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

Optional Extensions

<Ctrl> fixes center

<Shift> fixes figure relation



Interface FigureHandle

Methods

- `draw(Graphics g)` draws handle
- `getCursor()` returns a cursor which fits to the cursor movements
- `getOwner()` returns a reference to the owner of this handle
- `getLocation()` returns the location of the handle
- `contains(int x, int y)` returns whether (x,y) is at the handle

Dragging Methods

- `startInteraction(int x, int y, MouseEvent e, Drawview v)`
  - called when mouse is pressed
- `dragInteraction(int x, int y, MouseEvent e, Drawview v)`
  - called when mouse is moved
- `stopInteraction(int x, int y, MouseEvent e, Drawview v)`
  - called when mouse is released

Mouse Events

- **Signature of mouse interaction methods**
  - startInteraction / dragInteraction / stopInteraction
    - x/y coordinates of the mouse
    - e mouse event (can be used to access modifier keys
    - v access to the draw view (and over v also to the model)

```
public void dragInteraction(int x, int y,
                    MouseEvent e, DrawView v) {

   if (e.isControlDown()) {
      // <Ctrl>+Key is pressed
   }
   if (e.isShiftDown()) {
      // <Shift>+Key is pressed
   }
}
```