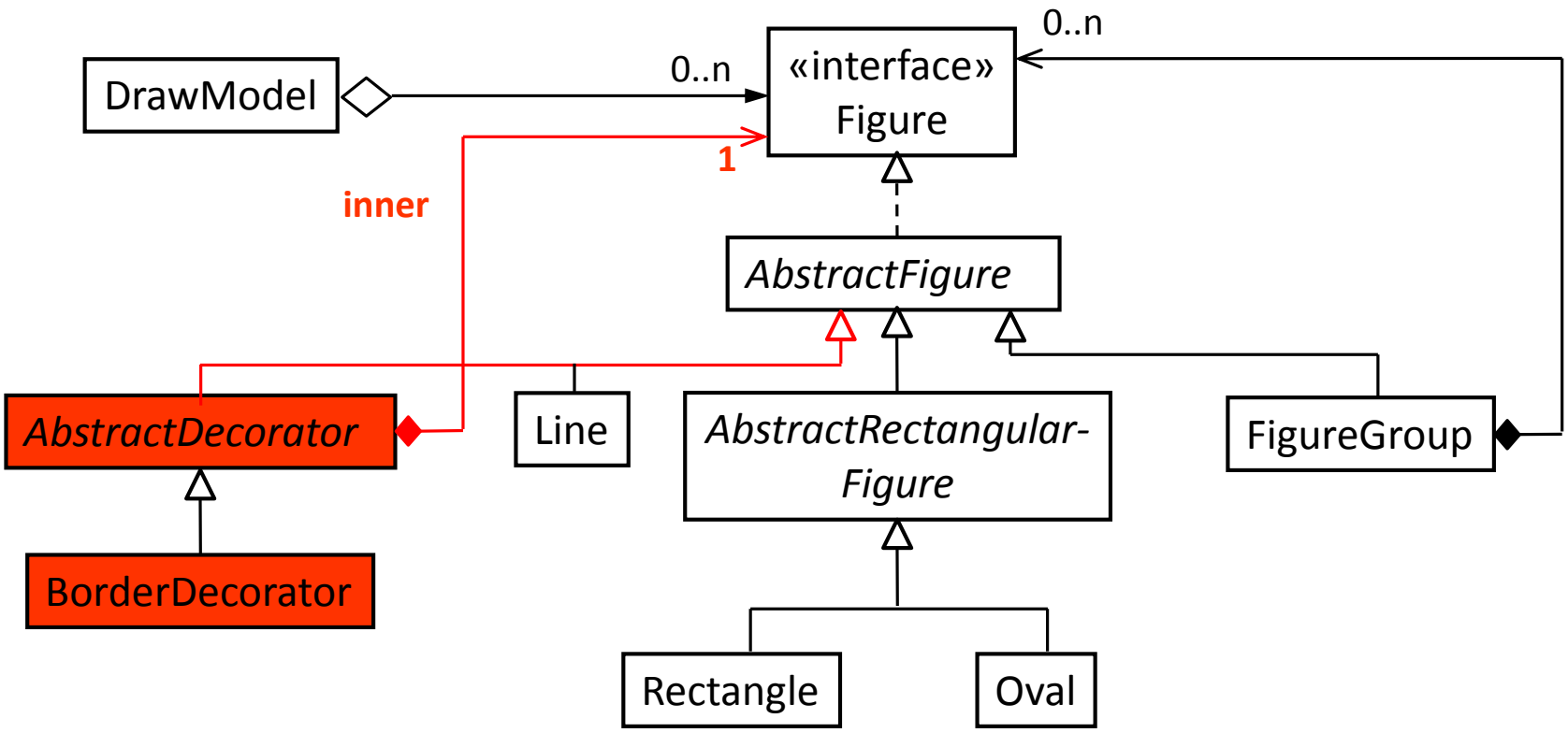
Assignment 9: Decorators



AbstractDecorator

```
public abstract class AbstractDecorator extends AbstractFigure {

   private Figure inner;

   private List<FigureHandle> handles;

   public AbstractDecorator(Figure figure) {
      this.inner = figure;
   }

   public Figure getDecoratedFigure() {
      return inner;
   }

   …
```

Cannot be declared final due to Java cloning

Only needed if handles are cached in the decorator

AbstractDecorator

```
@Override
public void draw(Graphics g) {
   inner.draw(g);
}

@Override
public AbstractDecorator clone() {
   try {
      AbstractDecorator f = (AbstractDecorator) super.clone();
      f.inner = (Figure) inner.clone();
      f.handles = null;
      return f;
   } catch (CloneNotSupportedException e){
      throw new InternalError(e);
   }
}

…
```

`draw`, `contains`, `move` etc. are all forwarded to the decorated figure

Decorated figure has to be cloned as well

Only needed if handles are cached

AbstractDecorator: getHandles

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

Problem: Handles refer to the wrong owner. As a consequence, the handles are not removed if the figure is removed.

```
public List<FigureHandle> getHandles() {
   List<FigureHandle> handles = new LinkedList<>();
   for(FigureHandle h: inner.getHandles()) {
      handles.add(new HandleDecorator(h));
   }
   return Collections.unmodifiableList(handles);
}

…
```

This handle-decorator returns the correct owner.

This implementation does not cache created handles, i.e. for every invocation a new list is created. Most likely, the handles are cached in the decorated figure.

AbstractDecorator.HandleDecorator

```
private final class HandleDecorator implements FigureHandle  {
   private final FigureHandle inner;

   public HandleDecorator(FigureHandle handle) {
      this.inner = handle;
   }

   @Override
   public Figure getOwner() {
      return AbstractDecorator.this;
   }

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

Class `HandleDecorator` is declared as inner class of class `AbstractDecorator`

Returns correct owner

The remaining methods are all forwarded to the original figure handle.

BorderDecorator

```
public class BorderDecorator extends AbstractDecorator {
   private static final int BORDER_OFFSET = 5;

   public BorderDecorator(Figure figure) {
      super(figure);
   }

   @Override
   public Rectangle getBounds() {
      Rectangle r = inner.getBounds();
      r.grow(BORDER_OFFSET, BORDER_OFFSET);
      return r;
   }

   …
```

Bounding box is enlarged

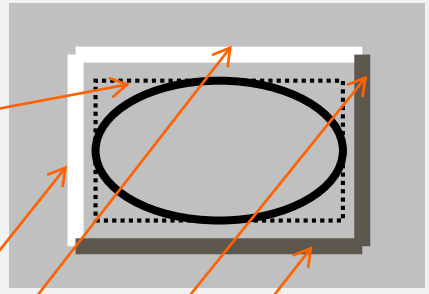
BorderDecorator

```
…

@Override
public void draw(Graphics g) {
   super.draw(g);
   Rectangle r = getBounds();
   g.setColor(Color.white);
   g.drawLine(r.x, r.y, r.x, r.y + r.height);
   g.drawLine(r.x, r.y, r.x + r.width, r.y);
   g.setColor(Color.gray);
   g.drawLine(r.x + r.width, r.y, r.x + r.width, r.y + r.height);
   g.drawLine(r.x, r.y + r.height, r.x + r.width, r.y+ r.height);
}

…
```

Returns enlarged bounds



BorderDecorator

```
…

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

@Override
public List<FigureHandle> getHandles() {
   …
}
```

Makes the figure selectable inside the border

Problem: the inherited handles are positioned inside the border. For a snap-to-grid it would be helpful, if the handles are placed on the border.
=> Would require new handles, but this may restrict the functionality of the figure (e.g. a polygon with unique handles)

Add Border Decorator Menu-Item

```
private JMenuItem createAddBorderDecoratorMenuItem() {
   JMenuItem m = new JMenuItem("Add Border Decorator");
   m.addActionListener( e -> {
      List<Figure> s = getSelection();
      clearSelection();
      for (Figure f : s) {
         BorderDecorator dec = new BorderDecorator(f);
         getModel().removeFigure(f);
         getModel().addFigure(dec);
         getView().addToSelection(dec);
      }
   });
   return m;
}
```