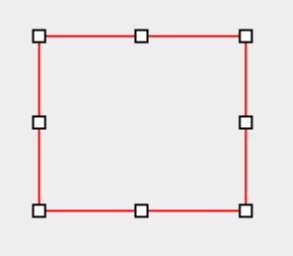
Hinweise zu Übung 4



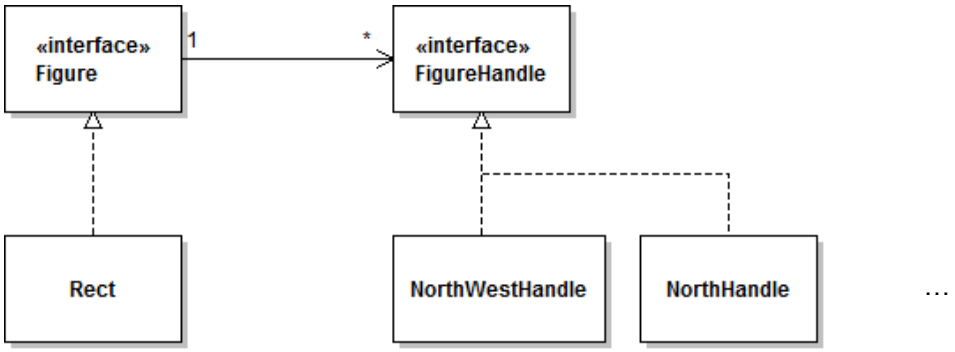
Auch für diese Übung haben wir wiederum einige Anregungen zusammengestellt, wie Sie die Aufgabe angehen können, damit ihre Figuren über Handles verändert werden können (die kleinen Quadrate in der nebenstehenden Figur). Jede Figur wird eine unterschiedliche Anzahl von Handles haben. Eine Linie hat typischerweise zwei Handles, ein Rechteck acht und ein Oval vier oder acht.

A. Welche Klassen braucht es?

Das folgende Diagramm zeigt Ihnen die Situation, wenn Sie eine Figur (z.B. die Klasse *Rect*) mit Handles erweitern wollen. Folgende Klassen müssen bereitgestellt oder geändert werden:

1. **FigureHandle-Klassen**
   Am einfachsten programmieren Sie für jeden Handle eine eigene Klasse. Das Diagramm unten beschränkt sich aus Platzgründen auf zwei solche Klassen: Der Handle oben in der Mitte (*NorthHandle*) und der Handle oben links (*NorthWestHandle*).

2. **Figuren-Klasse**
   Damit *JDraw* ihre Handles auch anzeigt, müssen diese in der Methode *getHandles* der jeweiligen Figur zurückgegeben werden.



B. Mögliches Vorgehen

Hier beschreiben wie Ihnen einige Schritte, denen Sie folgen können, um sich in diese Übung einzuarbeiten.

1. **Handle Klasse programmieren**
   Erzeugen Sie (z.B. auch im Paket *jdraw.figures*) die Klasse *NorthWestHandle* und implementieren Sie in dieser Klasse das Interface *FigureHandle*. Sie können auch gleich die fehlenden Methoden von ihrer IDE generieren lassen. Diese Methoden können Sie Schritt für Schritt implementieren:

   Figure getOwner():
   Diese Methode soll die Figur zurückgeben, zu der diese Handle-Instanz gehört, also eigentlich der Besitzer (owner) dieses Handles. Am einfachsten ist es wenn sie diesen Owner als Feld in der Handle-Klasse definieren und über den Konstruktor setzen.

   Point getLocation():
   Diese Methode soll die Position des Zentrums des Handles angeben. Am einfachsten ist es, wenn Sie diese Position direkt über die Position des Owners beziehen. Für den *NorthWestHandle* ist dies gerade die Position der Bounding-Box.

   ```
   public Point getLocation() {
      return owner.getBounds().getLocation();
   }
   ```

   Für die anderen Handles muss diese Methode natürlich anders implementiert werden.

   void draw(Graphics g):
   Diese Methode soll den Handle zeichnen, und zwar an der von der Methode *getLocation* bestimmten Position. Die folgende Implementierung zeichnet ein weisses Quadrat der Grösse 6 Pixel um die Position (sinnvollerweise definieren Sie eine Konstante für die Grösse des Handles).

```
public void draw(Graphics g) {
    Point loc = getLocation();
    g.setColor(Color.WHITE); g.fillRect(loc.x - 3, loc.y - 3, 6, 6);
    g.setColor(Color.BLACK); g.drawRect(loc.x - 3, loc.y - 3, 6, 6);
}
```

boolean contains(int x, int y)
Die Methode *contains* gibt an, ob sich die Position (x,y) innerhalb des Handles befindet.

Cursor getCursor()
Diese Methode definiert das Aussehen des Cursors wenn die Figur mit diesem Handle verändert wird. Für den *NorthWestHandle* bietet sich folgende Implementierung an:

```
public Cursor getCursor() {
    return Cursor.getPredefinedCursor(Cursor.NW_RESIZE_CURSOR);
}
```

2. **Handle in der Figur erzeugen**
Die Methode *getHandles,* die in jeder Figurenklasse implementiert wird, muss eine Liste von Handles zurückgeben. In der Figur *Rect* können Sie eine neue Instanz Ihres Handles in diese Liste einfügen. Die Methode sieht dann so aus:

```
public List<FigureHandle> getHandles() {
    List<FigureHandle> handles = new LinkedList<>();
    handles.add(new NorthWestHandle(this));
    return handles;
}
```

Wird nun in *JDraw* ein Rechteck gezeichnet und selektiert, dann erscheint ein Handle oben links. Fährt man mit der Maus über den Handle, dann sieht man einen speziellen Cursor. Aber die Figur kann über diesen Handle noch nicht verändert werden.

3. **Interaktion hinzufügen**
Wenn der Benutzer den Handle verschiebt, dann wird die Methode *startInteraction*, dann mehrmals die Methode *dragInteraction* und am Schlus die Methode *stopInteraction* aufgerufen. In der Methode *dragInteraction* wird der owner des Handles jeweils verändert. Die Methode *startInteraction* dient dazu, gewisse Variablen zu initialisieren, die dann in der Methode *stopInteraction* wieder aufgeräumt werden können. Der *NorthWestHandle* merkt sich bei Aufruf von *startInteraction* die Position der gegenüberliegenden – also der unteren rechten – Ecke der Figur, und in der Methode *stopInteraction* wird diese Referenz wieder gelöscht. Folgende Methoden erlauben dann, dass die Grösse der Figur über den Handle verändert werden kann.

```
private Point corner;

public void startInteraction(int x, int y, MouseEvent e, DrawView v) {
       Rectangle r = owner.getBounds();
       corner = new Point(r.x + r.width, r.y + r.height);
}

public void dragInteraction(int x, int y, MouseEvent e, DrawView v) {
       owner.setBounds(new Point(x, y), corner);
}

public void stopInteraction(int x, int y, MouseEvent e, DrawView v) {
       corner = null;
}
```

C. Ausfaktorisieren gemeinsamen Codes
Wenn Sie mehrere Handles implementiert haben, werden sie merken, dass einige Methoden immer wieder programmiert (oder eben kopiert) werden müssen. Wie in der letzten Übung führen Sie auch hier am einfachsten eine abstrakte Basisklasse ein (*AbstractFigureHandle*), welche die gemeinsamen Methoden enthält.

Wenn Sie den NorthWest-Handle über den rechten Rand hinaus bewegen, dann werden Sie feststellen, dass sich der Handle weiterhin an der Position oben links befindet, und auch der Cursor ändert sich nicht. Dieses Verhalten könnte mit dem State-Pattern gelöst werden.