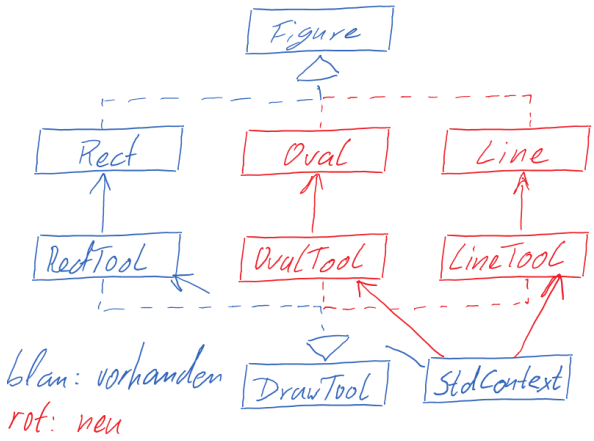
Hinweise zu Übung 3

Wir haben wiederum einige Anregungen zusammengestellt, wie Sie die Aufgabe angehen können. Sie brauchen sich aber nicht detailliert an diese Schritte zu halten, wenn Ihnen ein anderer Weg besser erscheint.

A. Welche Klassen braucht es?

Das folgende Bild zeigt Ihnen die Situation, wenn Sie *JDraw* mit den neuen Figuren *Oval* (oder Ellipse) und *Line* erweitern wollen. Für weitere / andere Figuren müssen Sie entsprechend weitere / andere Klassen dazu programmieren.

Sie brauchen keine vorhandenen Klassen abzuändern (mit der Ausnahme von *StdContext*, siehe Punkt 3. unten), müssen aber neue Klassen programmieren.



1. Figure-Klasse

Jede neue Figur braucht eine neue Klasse mit dem Code, der zum Zeichnen, Verschieben, Verändern der Grösse der Figur usw. benötigt wird. Diese Klasse muss das Interface *Figure* implementieren.

2. DrawTool-Klasse

Figuren werden durch jeweils ein Figuren-spezifisches *Tool-Objekt* ins Model eingefügt. Ist ein solches Tool aktiv, erhält es die Mouse-Events der Applikation und kann diese interpretieren. Diese Klasse muss das Interface *DrawTool* implementieren.

3. Registrierung Figur

Damit die neuen *DrawTools* dem System bekannt sind, müssen sie in der Klasse *StdContext* in der Methode *doRegisterDrawTools* registriert werden (Zeile 165, Position ist mit einem TODO markiert).

```
@Override
protected void doRegisterDrawTools() {
	// TODO Add new figure tools here
	DrawTool rectangleTool = new RectTool(this);
	addTool(rectangleTool);
}
```

B. Mögliches Vorgehen

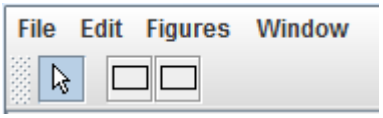
Hier beschreiben wir Ihnen einige Schritte, denen Sie folgen können, um sich systematisch in die entsprechenden Strukturen von *JDraw* einzuarbeiten.

1. Neues Tool erzeugen und registrieren

Kopieren Sie *RectTool.java* zu *OvalTool.java* und ändern Sie den Klassennamen entsprechend, sofern das Ihre IDE nicht ohnehin schon gemacht hat.

Registrieren Sie das *OvalTool* in *StdContext*, wie oben unter 3. beschrieben. Die Methode *doRegisterDrawTools* sieht dann wie folgt aus:

```
@Override
protected void doRegisterDrawTools() {
	DrawTool rectangleTool = new RectTool(this);
	addTool(rectangleTool);
	OvalTool ovalTool = new OvalTool(this);
	addTool(ovalTool);
}
```



Wenn Sie nun *JDraw* starten, sehen Sie das Rechteck-Tool zweimal angezeigt: Das bisherige und Ihr neues kopiertes. Da die Klassen bis auf den Namen identisch sind, verhalten sich beide Tools natürlich gleich und fügen die gleichen Rechteck-Figuren ein.

2. Tool verändern

Damit Sie rasch sehen können, dass tatsächlich zwei verschiedene Tool-Klassen verwendet werden, ändern Sie im *OvalTool* die Strings in den Methoden *activate*, *getIcon*, *mouseUp* und *getName*. Ersetzen Sie dort jeweils *Rectangle* durch *Oval* oder entsprechend. Eine Datei mit einem Icon *oval.png* ist bereits vorhanden. Sie können also diesen Namen einfach verwenden.

Wenn Sie nun *JDraw* starten, sollte die Schaltfläche, die das neue Tool repräsentiert, ein Oval zeigen. Wenn Sie das Tool aktivieren, sollte Sie auch die entsprechende Meldung in der Statuszeile sehen. Weiterhin werden aber natürlich die gleichen Rechtecke als Figuren eingefügt.

3. Neue Figuren-Klasse

Kopieren Sie *Rect.java* zu *Oval.java* und ändern Sie den Klassennamen entsprechend, sofern das Ihre IDE nicht ohnehin schon gemacht hat.

Ändern Sie in der Tool-Klasse *OvalTool* Typ (und sinnvollerweise Name) der Instanzvariable *newRect* und den Code in der Methode *mouseDown* so, dass eine Instanz der Klasse *Oval* erzeugt und dann benutzt wird.

Machen Sie auch noch eine einfache Änderung in der Methode *draw* der Klasse *Oval*. Zum Beispiel können Sie einfach eine andere Farbe verwenden.

Wenn Sie nun *JDraw* starten, sehen Sie, dass mit dem neuen Tool Ihre neue Figuren-Klasse erzeugt wird.

4. Figuren-Klasse fertig programmieren

Damit die *Oval*-Klasse richtig funktioniert, müssen Sie einige Methoden anpassen. Klar anders verhalten sollten sich vor allem die Methoden *draw* und *contains*. Die erste soll ein Oval zeichnen, die zweite nur dann *true* zurückgeben, wenn der Punkt *(x, y)* wirklich innerhalb der Ellipse liegt und nicht nur innerhalb der Bounding Box.

Sie können hier auf verschiedene Arten weiterarbeiten. Eine Möglichkeit ist, einfach weiterhin die Bounding Box zu speichern und in *contains* entsprechende Berechnungen zu programmieren. Eine andere Möglichkeit ist, auf den Datentyp *java.awt.geom.Ellipse2D.Double* zu wechseln. Der bietet die Berechnung für *contains* bereits fertig an. Dafür braucht es sonst noch ein paar Änderungen in verschiedenen Methoden Ihrer Klasse *Oval*.

Es gibt in der Klasse *Oval* auch eine Menge von Kommentaren, die Sie aus *Rect* kopiert haben und vielleicht ebenfalls anpassen möchten.

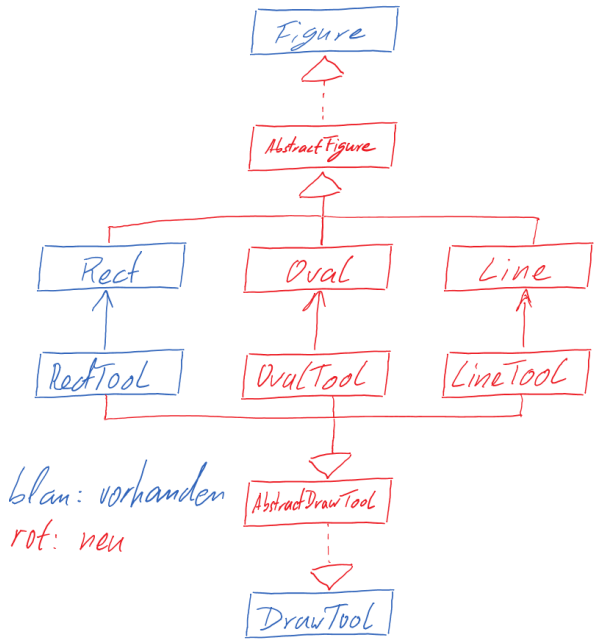
5. Weitere Figur(en)

Sie haben die wichtigsten Schritte auf dem Weg zu eigenen Figuren ausprobiert und gesehen, worauf es ankommt. Nun können Sie weitere Figuren hinzufügen. Etwas schwieriger als ein Oval ist es, eine Linie zu programmieren. Die Schwierigkeiten werden Ihnen aber vermutlich erst auffallen, wenn Sie Linien in verschiedenen Richtungen zu erzeugen versuchen. Für Linien haben wir Ihnen ebenfalls bereits eine Icon-Datei mitgeliefert: *line.png*.

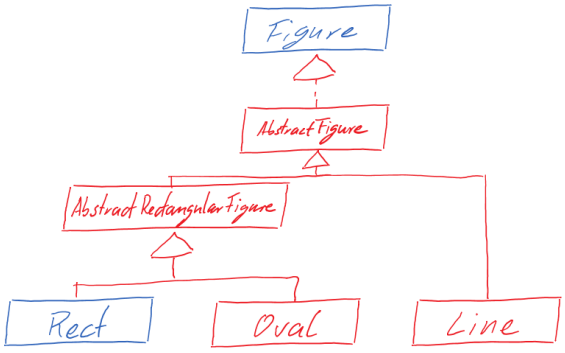
Natürlich können Sie auch ganz andere spannende Figuren programmieren. Lassen Sie Ihrer Fantasie freien Lauf!

C. Herausfaktorisieren gemeinsamen Codes

Wenn Sie mehrere Figuren implementiert haben dann werden sie merken, dass doch einige Methoden immer wieder programmiert (oder eben kopiert) werden. Diese werden am einfachsten in eine abstrakte Basisklasse ausgelagert (*AbstractFigure* bzw. *AbstractDrawTool*):



Vielleicht stellen Sie auch fest, dass der Art nach über Rechtecke definierte Figuren mehr Gemeinsamkeiten haben, als andere, wie Linien. Dann lohnt es vielleicht auch, eine kleine Hierarchie von Abstraktionen (und entsprechend abstrakten Klassen) zu designen:



Bei den Figuren bieten sich folgende Methoden an die in einer abstrakten Figurenklasse implementiert werden können:

- addFigureListener / removeFigureListener mit der Speicherung der Listener
- propagateFigureEvent um einen FigureEvent an alle Listener zu verschicken
- clone (kann vorerst null zurückgeben)
- getHandles (kann vorerst null zurückgeben)

Allenfalls können sie in der Basisklasse auch bereits eine Position oder ein Rechteck definieren um die Position / Grösse der Figur zu definieren. In diesem Fall können auch die Methoden move und setBounds bereits in der abstrakten Basisklasse implementiert werden.

Bei den DrawTools bieten sich folgende Methoden an:

- getName (basierend auf einem Namen der mit dem Konstruktor übergeben wird)
- getIcon (basierend auf einem Pfad/Name der mit dem Konstruktor übergeben wird)
- getCursor (Default-Cursor)
- activate (zeigt den Namen des Tools in der Statuszeile an)
- deactivate (löscht den Inhalt der Statuszeile)
- mouseDown/mouseDrag/mouseUp