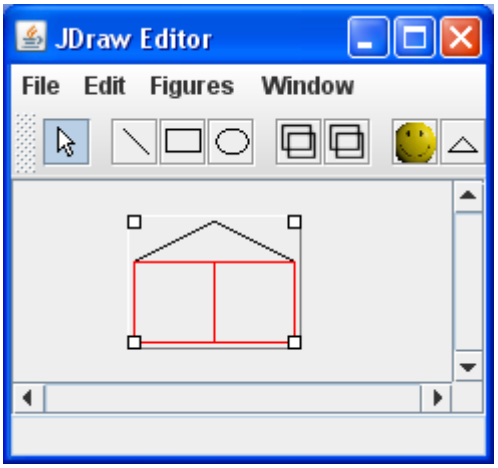
Arbeitsblatt Decorator

Sie haben das Decorator-Pattern kennengelernt. In dieser Aufgabe sollen spezielle Aspekte der Decoratoren untersucht werden. Wenn Sie die beschriebenen Probleme reproduzieren wollen, können Sie entweder einen Decorator verwenden, den Sie bereits für die Übung programmiert haben, oder Sie können den letzte Woche in der Vorlesung programmierten Green-Decorator verwenden, den Sie auf dem AD finden.

Gruppenfiguren

Erzeugen Sie eine Gruppenfigur und dekorieren Sie diese mit einem Decorator. Das folgende Bild zeigt eine mit einem Rahmen-Decorator dekorierte Gruppenfigur.



Wenn Sie nun auf dieser Selektion das *Ungroup* aufrufen, wird die Gruppenfigur nicht in die Teilbilder zerlegt.

Wenn Sie eine Figur mit dem Green-Decorator versehen und dann zusätzlich noch mit einem Rahmen-Decorator, erwarten Sie dann, dass Sie auf dieser Figur den Green-Decorator wieder entfernen können ohne dass der Rahmen verschwindet?

Aufgaben, Vorgehen:

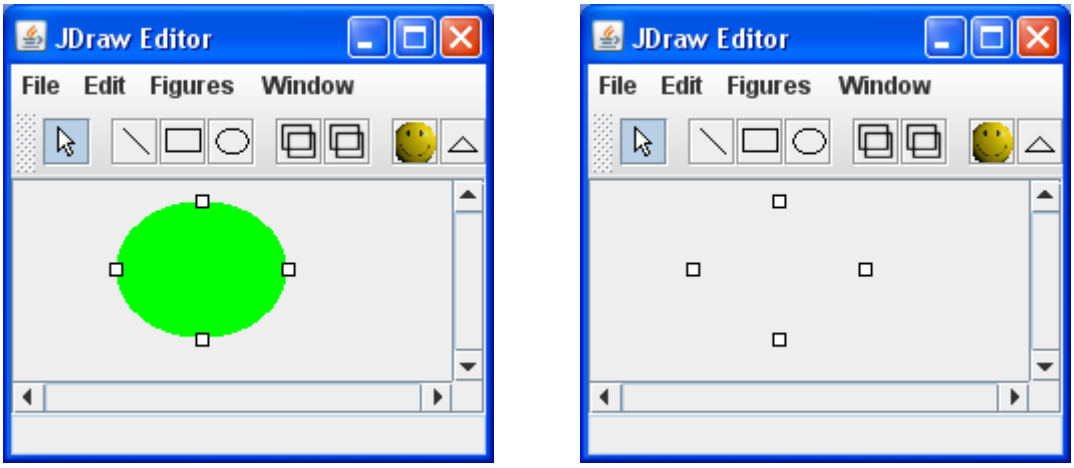
1. Problemanalyse (alleine oder zu zweit) : 10 Minuten
   - Überlegen Sie sich, warum die Gruppenfigur nicht ausgepackt wird.
   - Schlagen Sie eine Lösung vor, die es erlauben würde, die dekorierte Gruppenfigur auszupacken, oder zumindest auf die gruppenspezifischen Funktionen (wie z.B. ein `getNumberOfFigures`) zugreifen zu können bzw. eine Lösung die es erlauben würde, den Green-Decorator zu entfernen ohne den Rahmen-Decorator zu entfernen.
   - Beschreiben Sie auch, was für ein Resultat Sie nach einem Ungroup auf dem Bildschirm erwarten.

2. Expertenrunde: 15 Minuten
   Fassen Sie Ihre Beobachtungen zusammen.
   Letzte Woche haben wir den Decorator mit Inheritance verglichen. Vergleichen Sie diese beiden Lösungsvarianten nun bezüglich dieses Aspekts.

3. Präsentation:
   Sie haben ca. 5 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 4 Abschnitte:
   a) Analyse des Problems (und allenfalls Verallgemeinerung des Problems)
   b) Problemlösung (für die Gruppenfigur mit Beschreibung des Verhaltens)
   c) Lösung des generellen Problems
   d) Vergleich Decorator - Inheritance anhand dieses Aspektes

Löschen von Decoratoren

Dekorieren Sie eine Figur und löschen Sie diese anschliessend. Sie werden dann folgendes Bild sehen:



Löscht man eine Figur, wird sie aus dem Modell entfernt. Alle Views werden entsprechend informiert und entfernen die gelöschte Figur mit der Methode `removeFromSelection` aus ihrer Selektion. Aus Effizienzgründen werden alle angezeigten Handles in einer separaten Liste gehalten. Die Handles einer gelöschten Figur müssen aus dieser Liste entfernt werden.

```
@Override
public void removeFromSelection(Figure f) {
	if (selection.remove(f)) {
		handles.removeIf(h -> h.getOwner() == f);
	}
}
```

Obwohl dieser Code ausgeführt wird werden die Handles nicht gelöscht.

Aufgaben, Vorgehen:

1. Problemanalyse (alleine oder zu zweit): 10 Minuten
   - Überlegen Sie sich, warum die Handles in dieser Situation nicht gelöscht werden.
   - Schlagen Sie eine Lösung vor, wie der Decorator korrigiert werden muss damit obiger Code wieder richtig funktioniert

2. Expertenrunde: 15 Minuten
   Fassen Sie Ihre Beobachtungen zusammen, und halten Sie fest, wie der Decorator korrigiert werden muss. Vielleicht haben Sie verschiedene Lösungen in die Expertenrunde eingebracht. Vergleichen Sie diese Lösungsvarianten.

3. Präsentation:
   Sie haben ca. 5 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 2 Abschnitte:
   a) Analyse des Problems
   b) Problemlösung

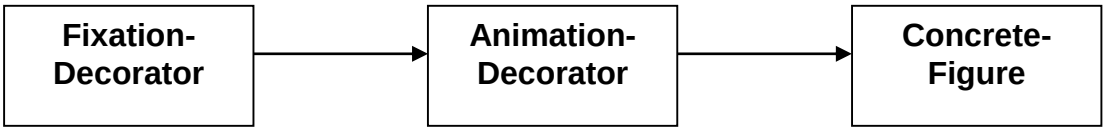
Animator

Für diese Frage nehmen wir an, dass wir einen `AnimationDecorator` programmiert hätten. Dieser startet einen eigenen Thread und ruft regelmässig die `move`-Methode der dekorierten Figur auf.

Nehmen wir nun an, dass wir zusätzlich noch einen `FixationDecorator` implementiert haben, der verhindert, dass die Figur mit den Methoden `move` und `setBounds` oder über die Handles verändert wird. Aufrufe der Methoden `move` und `setBounds` werden in diesem Decorator nicht an die innere Figur weitergeleitet, d.h. sie sind wie folgt implementiert:

```
@Override public void move(int dx, int dy) { }
@Override public void setBounds(Point origin, Point corner) { }
```

Wenn Sie nun eine Figur mit dem `AnimationDecorator` dekorieren und anschliessend mit dem `FixationDecorator`, so "wandert" die Figur weiterhin umher.



Aufgaben, Vorgehen:

1. Problemanalyse (alleine oder zu zweit): 10 Minuten
   - Überlegen Sie sich, warum der `FixationDecorator` in diesem Fall seine Aufgabe nicht wahrnimmt.
   - Wie sieht es aus, wenn der Animator als Unterklasse der konkreten Figur definiert wird und der Fixator als Unterklasse des Animators?
   - Schlagen Sie eine Lösung vor, die es erlauben würde, dass der `FixationDecorator` wie erwartet arbeiten kann, auch wenn er nach dem `AnimationDecorator` hinzugefügt wird.

2. Expertenrunde: 15 Minuten
   Fassen Sie Ihre Beobachtungen zusammen.
   Letzte Woche haben wir den Decorator mit Inheritance verglichen. Vergleichen Sie diese beiden Lösungsvarianten nun bezüglich dieses Aspekts.

3. Präsentation:
   Sie haben ca. 5 Minuten Zeit, um Ihre Resultate zu präsentieren. Eine kurze Diskussion mit der Klasse rundet Ihre Präsentation ab.
   Gliedern Sie Ihre Präsentation in die 3 Abschnitte:
   a) Analyse des Problems (und allenfalls Verallgemeinerung des Problems)
   b) Problemlösung
   c) Vergleich Decorator - Inheritance anhand dieses Aspektes