Übung 09: Design Patterns

Abgabetermin: 1. 6. 2017, 8:15

Name: _____ **Matrikelnummer**: _____

Informatik: ☐ G1 (Marr) ☐ G2 (Prähofer) ☐ G3 (Prähofer) ☐ G4 (Löberbauer)

WIN: ☐ G1 (Khalil) ☐ G2 (Hummel) ☐ G3 (Khalil)

Aufgabe	Punkte	abzugeben schriftlich	abzugeben elektronisch	korr.	Punkte
Übung 9	24	Java-Programm	Java-Programm	☐	

Übung 9: MicroDraw

MicroDraw ist eine einfache GUI-Applikation, mit der man Graphiken aus primitiven Shapes erstellen kann. Abbildung 1 zeigt die Applikation.

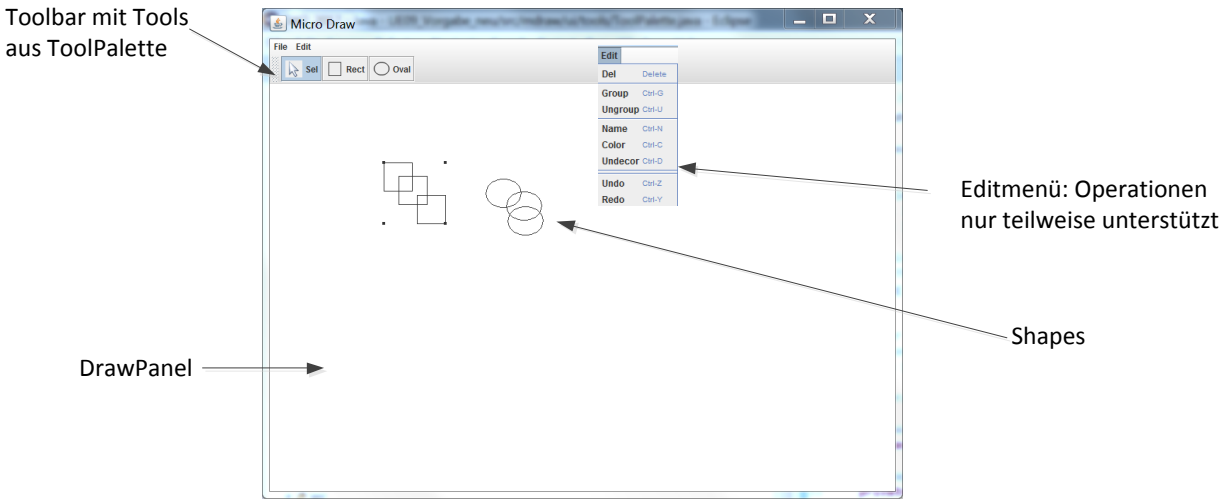

Abbildung 1: Screenshot von MicroDraw

Die Applikation hat eine Toolbar mit Tools zum Anfügen von Rechtecken und Ovalen und ein Selektionstool. Mit dem Selektionstool kann man Elemente selektieren und Gruppen von selektierten Objekten bilden (Menüeintrag „Group“).

Die Implementierung der Applikation hat eine Architektur wie in Abbildung 2 gezeigt.

- Die angezeigten graphischen Objekte sind im Package `mdraw.shapes` implementiert. Sie sind alle von Interface `Shape` abgeleitet, welches Methoden für Zugriff auf und Setzen von Position und Größe, Zeichnen und Füllen, Selektion und Erzeugen einer Kopie definiert. Es gibt konkrete `Shape`-Klassen `Rect`, `Oval` und `Group`. `Groups` sind nach dem Composite-Pattern gebildet.
- Klasse `ShapeModel` zusammen mit Listeners und Event-Objekten aus dem Package `mdraw.model` implementiert ein Modell nach dem MVC-Prinzip. `ShapeModel` verwaltet eine Reihe von Shapes und eine Menge von aktuell selektierten Shapes. Es benachrichtigt über Änderungen in der Menge der Shapes mit `shapeChanged`-Ereignissen und Änderungen bei der Selektion mit `shapeSelection`-Ereignissen.
- `ShapeApp` und `ShapePanel` implementieren die graphische Anzeige. Sie horchen auf `shapeChanged`- und `shapeSelection`-Ereignisse des Modells. In `ShapeApp` gibt es eine Reihe von `Action`-Implementierungen, die die Reaktionen auf die Menüoperationen realisieren.

- Wesentlich bei der Applikation ist die Behandlung der Mouse-Ereignisse im `ShapePanel`. Die Reaktion auf die Mouse-Ereignisse ist abhängig vom im Toolbar ausgewählten Tool. Das `ShapePanel` leitet daher alle Mouse-Ereignisse an das in der Toolbar ausgewählte Tool weiter. Für die Implementierung von Tools gibt es eine abstrakte Basisklasse `Tool` (im Package `mdraw.tools`). `Tool` leitet von `AbstractAction` ab (kann also als `Action` verwendet werden) und implementiert `MouseListener`. Es gibt drei konkrete Tools: RectTool und OvalTool erzeugen bei MouseClicks Rechtecke und Ovale. Das SelTool erlaubt die Selektion von Elementen (die dann gruppiert werden können). Die Verwaltung der Tools erfolgt in der `ToolPalette`, die direkt als Toolbar verwendet wird (erweitert `JToolBar`). Man kann der `ToolPalette` Tools hinzufügen, die dann in der Toolbar der Applikation erscheinen.

Sie bekommen die Implementierung dieser Applikation als Download zur Verfügung gestellt.

Anmerkung: Die Applikation wurde gegenüber der in der LVA gezeigten Variante vereinfacht und bietet nun keine Möglichkeiten zum Verschieben und Vergrößern von Shapes.

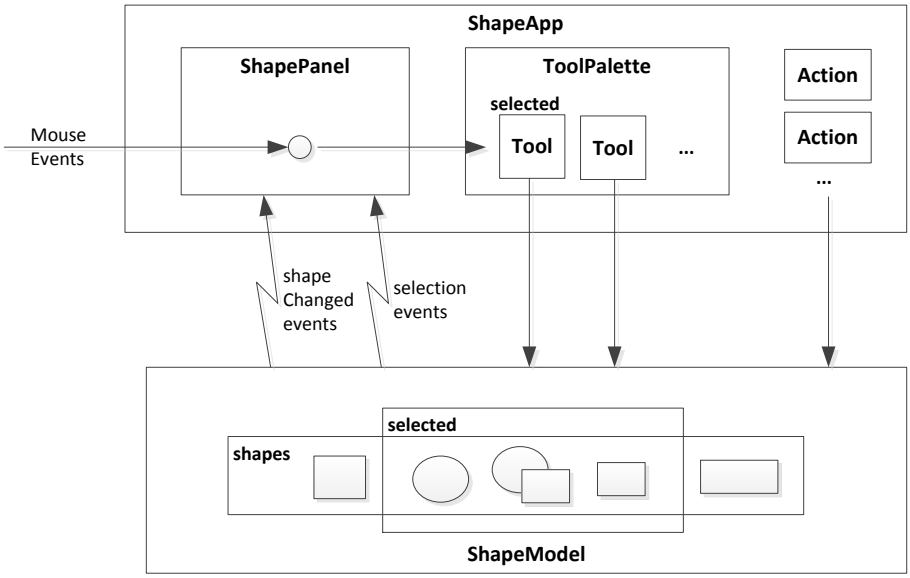

Abbildung 2: Architektur

Erweiterungen

Der Projektleiter möchte nun, dass Sie die Applikation im Funktionsumfang erweitern. Unser Projektleiter möchte, dass Sie dabei Design Patterns anwenden und formuliert die Aufgabe folgend:

Decorator (9 Punkte)

Erweitern Sie das Klassensystem für Shapes so, dass die Shapes mit Decorators dekoriert werden können. Implementieren Sie zuerst allgemein das Design Pattern *Decorator* für Shapes und realisieren Sie dann die folgenden konkreten Decorators:

(1) Mit einem `NameDecorator` soll einem ausgewählten Shape ein Name gegeben werden können (der dann natürlich auch angezeigt wird). Mit dem Menüeintrag *Name* (siehe Abbildung 1) soll dem aktuell selektierten Shape ein `NameDecorator` angefügt werden (es darf dazu nur ein Objekt selektiert sein). Der Name soll durch einen `JOptionPane`-Dialog wie folgt eingelesen werden:

```
String name = JOptionPane.showInputDialog(frame, "Input name for decoration",
                    "Name", JOptionPane.QUESTION_MESSAGE);
```

(2) Mit einem `ColorDecorator` sollen aktuell ausgewählte Shapes (können mehrere sein) mit einer Füllfarbe dekoriert werden können. Die Farbe soll über einen `JColorChooser` ausgewählt werden:

```
Color color = JColorChooser.showDialog(frame, "Color", Color.WHITE);
```

Verwenden Sie Methode `fill(Graphics g, Color g)` von `Shape`, um die Füllung zu zeichnen.

(3) Für ein dekoriertes Shape soll der Decorator auch wieder entfernt werden können (`undecorate`).

Die Menüeinträge und auch die `Action`-Objekte sind in der Applikation schon vorbereitet. Siehe dazu Menüeinträge *Name*, *Color* und *Undecor* und `Action`-Objekte `nameAction`, `colorAction` und `undecorateAction` in Klasse `DrawApp` (die `actionPerformed`-Methoden der `Action`-Objekte werfen zurzeit eine `UnsupportedOperation-Exception`).

Hinweis: Beachten Sie, dass um ein Shape zu dekorieren das Shape zuerst aus dem Modell entfernt, dieses dann dekoriert, und schließlich das dekorierte Shape wieder eingefügt werden muss. Bei `undecorate` ist genau der umgekehrte Vorgang anzuwenden.

Command- und Singleton-Patterns (15 Punkte)

Es soll nun die Applikation so erweitert werden, dass Undo/Redo von Operationen ermöglicht werden. Implementieren Sie nach dem Command-Pattern Undo/Redo aller Operationen, die die Shapes im `ShapeModel` verändern.

Hinweis: Änderungen von Selektionen oder Operationen, die keine Veränderungen bei den graphischen Objekten bewirken sollen nicht *undoable* sein!!

Für Undo/Redo gibt es in der Klasse `DrawApp` bereits die entsprechenden Menüoperationen und die `Action`-Objekte `undoAction` und `redoAction` (die zurzeit eine `UnsupportedOperation-Exception` werfen).

Folgend sind Codestellen aufgelistet, die Veränderungen beim Modell bewirken, d.h., hier sind Aufrufe beim `ShapeModel` zu finden, die das Modell verändern, wie z.B. `model.addShape(shape)`. Diese Aufrufe sind zur Unterstützung von Undo/Redo entsprechend anzupassen:

- Anfügen von Rechtecken bei `RectTool` und Anfügen von Ovals bei `OvalTool`
- Löschen von Shapes in `Action`-Objekt `delAction` in `DrawApp`
- `Action`-Objekte `nameAction`, `colorAction` und `undecorAction` für Decorators in `DrawApp`
- `Action`-Objekte `groupAction` und `ungroupAction` in `DrawApp` für Groups

Bei der Umsetzung der Undo/Redo von Operationen nach dem Command-Pattern brauchen Sie einen globalen Zugriff auf den Command-Handler, der ein Singleton in der Applikation ist. Realisieren Sie diesen Zugriff mittels des *Singleton*-Patterns.