Institut für Informatik
Prof. Dr. Oliver Vornberger
Lukas Kalbertodt, B.Sc.

Universität Osnabrück, 05.04.2016
`http://www-lehre.inf.uos.de/~cg`
Testat bis 13.04.2016, 14:00 Uhr

Übungen zu Computergrafik

Sommersemester 2016

Blatt 1: MVC

Übungsbetrieb

In der Vorlesung am Dienstag wird ein Aufgabenblatt verteilt, das bis einschließlich Mittwoch (letztes Testat um 13:30 Uhr) der darauffolgenden Woche zu bearbeiten ist. Die Aufgabenblätter finden sich auch auf der Veranstaltungswebseite `www-lehre.inf.uos.de/~cg/2016`.

Die Übungen finden donnerstags von 10:15 - 11:45 Uhr in Raum 93/E33 und von 12:15 - 13:45 Uhr in Raum 93/E06 statt. In den Übungen wird das neue Aufgabenblatt durchgesprochen und die Lösung zum alten Aufgabenblatt diskutiert.

Testatbetrieb

Begleitend zur Veranstaltung finden wöchentliche, 30-minütige Testate bei den Tutoren der Veranstaltung statt. Das aktuelle Aufgabenblatt ist bis zum Testattermin zu bearbeiten und dem jeweiligen Tutor zum entsprechenden Termin in Raum 93/E10 vorzulegen sowie in das unten genannte Web-System hochzuladen.

Achten Sie bei der Bearbeitung der Aufgaben darauf, diese *vor* dem Testat einmal auf dem Zielrechner getestet zu haben und außerdem natürlich auf sinnvolle Kommentierung mit JavaDoc.

Die Testate erfolgen in Zweierteams, zu denen man sich von Donnerstag, 07.04. 9:00 Uhr, bis Sonntag, 10.04. 18:00 Uhr, über eine Webapplikation anmelden kann: `https://cg-testate.informatik.uni-osnabrueck.de` Verwenden Sie Ihren Rechenzentrums-Login.

Piazza

Unter `piazza.com/uni-osnabrueck.de/spring2016/cg16/home` finden Sie eine Instanz des Q&A-Systems Piazza. Dort können Sie Fragen und Probleme mit uns und Ihren Kommilitonen diskutieren. Außerdem werden dort Anmerkungen zu den Übungsblättern und sonstige Ankündigungen gemacht. Alle bis zum 04.04.2014, 18 Uhr in StudIP eingetragenen Studenten werden automatisch eingeladen. Sollten Sie nicht eingeladen worden sein, müssen Sie sich selbst registrieren.

Scheinvergabe

Um die Zulassung zur Klausur zu erhalten, müssen alle bis auf eins der ausgegebenen Übungsblätter erfolgreich bearbeitet (mindestens 50% der Punkte) und dem Tutor vorgeführt werden. Zum Abschluss der Veranstaltung entscheidet eine Klausur über die Scheinvergabe.

Aufgabe 1.1: Model-View-Controller (50 Punkte)

Melden Sie sich an einem Rechner in 93/E10 an und legen Sie die im Anhang befindliche Datei `mvcbeispiel.jar` in Ihr Arbeitsverzeichnis.

Mit dem Befehl `jar xvf mvcbeispiel.jar` wird das Archiv entpackt. Der Befehl `java -jar mvcbeispiel.jar` startet die Applikation.

Importieren Sie die Applikation auch in Ihr Eclipse-Arbeitsverzeichnis, so dass Sie den Quelltext dort bearbeiten und Ihrem Tutor im Testat vorführen können.

Erklären Sie Ihrem Tutor den Aufbau der Applikation und anhand dessen die allgemeine Idee hinter dem Model-View-Controller-Entwurfsmuster.

Die Applikation präsentiert beim Start lediglich eine funktionslose View. Implementieren Sie eine Variante des Model-View-Controller-Entwurfsmuster mit Hilfe des Observer/Observable-Design-Patterns. Wie in der Vorlesung vorgestellt, *beobachtet* die View das Model und wird benachrichtigt, wenn sich dort etwas geändert hat.

Für diesen Aufgabenteil reicht es, wenn Sie die beiden Buttons mit Funktionalität versehen, so dass sich bei einem Klick auf den *Erhöhe*-Button der Wert im Model um 1 erhöht, beim *Verringern* um 1 erniedrigt. Im `JLabel` in der Mitte soll immer der aktuelle Wert angezeigt werden.

Ergänzen Sie außerdem eine geeignete (doppelte) Fehlerbehandlung, die dafür sorgt, dass sich der Zahlwert immer nur im Intervall von 0 bis 100 bewegen kann.

Musterlösung:

Model View Controller

Das Model-View-Controller Pattern ist ein sehr weit verbreitetes Konzept zur Strukturierung von Softwareprogrammen. Die grundsätzliche Idee dabei besteht darin, den Aufbau eines Programms komponentenweise in drei Teile zu trennen: Eben Model, View und Controller. Ziel dabei ist es, einzelne Teile des Codes möglichst wiederverwertbar gestalten zu können. Dies soll dazu führen, dass man Programme einfacher erweitern oder ändern kann.

Model

Im Model werden alle Daten einer Applikation abgelegt und Teile der grundlegenden Manipulationslogik. Im vorgegebenen Beispiel wird dort die aktuell gewählte Zahl vorgehalten. Um das Model stets in einem sinnvollen Zustand zu halten, ist es sinnvoll Fehlerprüfung in den Manipulationsmethoden (hier beispielsweise setWert()) einzusetzen. Das Model enthält im allgemeinen keine reine Applikationslogik, es kann jedoch sinnvoll sein, dass beispielsweise grafische Objekte, wie eine Linie, neben ihrem Zustand (also beispielsweise den Eckpunkten und ihrer Farbe) auch weiß wie sie sich auf eine gegebene Oberfläche zeichnen kann. Dazu mehr auf dem nächsten Aufgabenblatt.

View

View-Objekte sind für die Darstellung von Daten verantwortlich und können Möglichkeit bieten, Daten durch den Benutzer zu ändern oder anzupassen. Die Daten selbst sind jedoch wie oben beschrieben im Model gespeichert. Um auch View-Objekte möglichst wiederverwertbar zu gestalten, ist es ratsam, diese in generischer Weise konfigurierbar zu machen. In der gegebenen Applikation ist in der Klasse ZaehlerView im Grunde ein Aussagen und Funktionsfreies JPanel konfiguriert, das grundsätzlich auch in jeder anderen Applikation mit anderm Zweck, als hier verwendet, zum Einsatz kommen könnte. Das gegebene Beispiel ist natürlich sehr akademisch, man kann sich vielleicht besser ein Fenster für eine Betriebssystem-Benachrichtigung vorstellen. Dieses zeigt immer einen Text und eine Menge von Buttons, welche Funktion das Fenster aber genau hat, muss in der View gar nicht definiert sein. Dies erledigt der Controller.

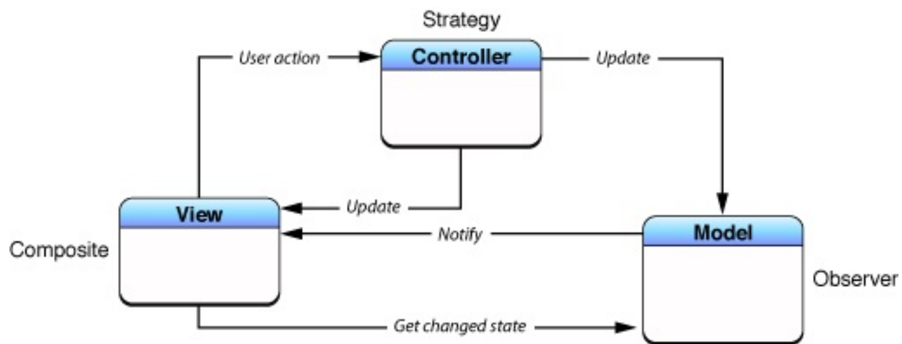
Controller

Für die Verbindung zwischen den möglichst generischen Views und den spezialisierten Model-Klassen sind die Controller verantwortlich. Im Allgemeinen kann man sagen, dass sie dafür sorgen, dass die Operationen, die ein User in einer View auslöst, an das entsprechende Model propagiert werden. Außerdem kann der Controller genutzt werden, um die View und das Model initial aufzubauen und miteinander bekannt zu machen. Im gegebenen Beispiel legt der Controller die für seinen Zweck nötige View an und konfiguriert diese - genau wie das entsprechende Model. Schließlich enthält der Controller den größten Teil der eigentlichen Applikationslogik, also den Teil der des Programmcodes, der die eigentlich geforderten Dinge tut. In der Beispielsapplikation legt er also alle nötigen Listener an und implementiert deren Logik: Was soll passieren, wenn ein Nutzer auf einen Button klickt? Was soll passieren, wenn ein Nutzer am Slider zieht?

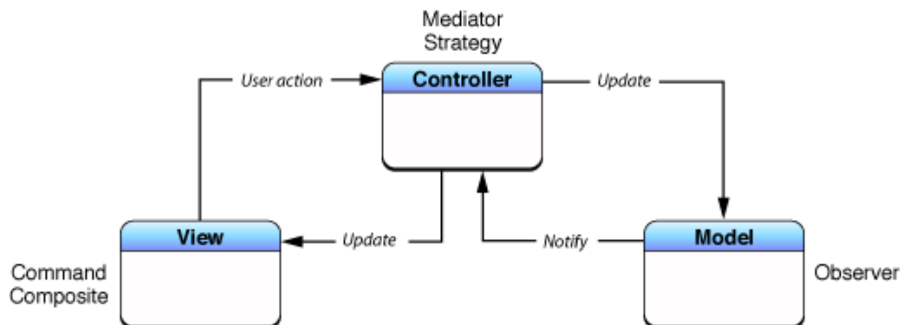
Zusammenspiel

Während die Grenzen für die Zuständigkeit der einzelnen Teile des MVC-Patterns relativ deutlich abgegrenzt sind, gibt es für das Zusammenspiel im Wesentlichen zwei häufig eingesetzte Varianten.

In der ersten Variante kommuniziert die View direkt mit dem Model. In der Aufgabe sollte hierzu das Observer/Observable-Pattern eingesetzt werden. Dabei erweitert eine Klasse die Systemklasse Observable und weitere Klassen, die das Interface Observer implementieren, können das Observable nun beobachten. Das Observable, hier das Model, wird im ersten Fall also von der View beobachtet und benachrichtigt diese per `stateChanged` und `notfiyObservers` über entsprechende Änderungen. Die View implementiert nun die Methode `update`, um sich entsprechend selbst zu ändern, wenn das Model seinen Zustand geändert hat:



In der zweiten Variante kommuniziert die View überhaupt nicht direkt mit dem Model. Das Anpassen der View an den neuen Zustand erfolgt über den Controller, der diesmal der `Observer` ist und die View entsprechend anpasst:



Während beide Vorgehen zu einer sinnvollen Code-Strukturierung beitragen, hat letztere Methode den weiteren großen Vorteil, dass Model und View sich niemals direkt kennen müssen, so dass die View noch generischer gehalten werden kann. Sie muss also nicht wissen, welche Werte aus dem Model abgeholt werden müssen, um sich selbst anzupassen.

Für den Code siehe Anhang Aufgabe 3.

Aufgabe 1.2: Slider und Eingabefeld (20 Punkte)

Ergänzen Sie die Applikation aus Aufgabe 1 dahingehend, dass sich die Zahl auch dann im Bereich von 0 bis 100 entsprechend ändert, wenn an dem `JSlider` gezogen wird.

Erweitern Sie die Applikation außerdem so, dass der Benutzer in einem Eingabefeld eine Zahl eingeben kann. Die Zahl soll nach Drücken der Eingabe-Taste am Model gesetzt werden und anschließend in der Anwendung im `JLabel` und auf dem `JSlider` angezeigt werden.

Achten Sie auf eine geeignete Fehlerbehandlung.

Musterlösung:

Siehe Anhang Aufgabe 3.

Aufgabe 1.3: Look and Feel (30 Punkte)

Erweitern Sie die Applikation um eine `JComboBox`, mit der das *Look and Feel* der Applikation umgestellt werden kann. Auswählbar sollen die Designs sein, die auf der Plattform, auf der das Programm läuft, zur Verfügung stehen.

Hinweis: Zum Umgang mit *Look and Feels* sind besonders die Klassen `UIManager` und `SwingUtilities` von Bedeutung. Machen Sie sich damit in der Java-Dokumentation vertraut.

Erstellen Sie ein eigenes JAR-File. Das JAR-File sollte ohne Angabe der Main-Klasse auf der Kommandozeile ausführbar sein.

Musterlösung:

Siehe Anhang.