Die Aufgaben müssen einzeln bearbeitet und abgegeben werden. Die Abgabe muss bis **spätestens Donnerstag 07.07.2011 um 23:59 Uhr** über unser Hausaufgabenabgabesystem **http://seblog.cs.uni-kassel.de/pmss11/** erfolgen. Die Abgabe ist nur als einzelne *.zip oder *.jar-Datei möglich. Daher müssen alle für eine Abgabe relevanten Daten zu einem solchen Archiv kombiniert werden. Diese Hausaufgabe gibt **29+6** Punkte.

Hinweise zur Abgabe:

- Die Abgabe **MUSS** als exportiertes Eclipse Projekt erfolgen. Falls Sie mehrere Projekte anlegen, können diese alle in eine .zip Datei gepackt werden (erledigt die Eclipse Export Funktion automatisch!). Sind die Projekte nicht korrekt exportiert, können diese bei der Korrektur nicht berücksichtigt werden (es bietet sich also an, den Import des exportierten Projekts auszuprobieren).

  WICHTIG Bennen Sie ihre Projekte nach folgendem Schema:

  ```
  PMSS2011_HA<a>_A<b>_<Matrikelnummer>,
  ```

  wobei <a> für die aktuelle Hausaufgabe und <b> für die Aufgabennummer steht. Beispiel:

  ```
  PMSS2011_HA6_A1_12345678.
  ```

Vorbereitung

Zur Bearbeitung dieser Aufgabe benötigen Sie ein fertig implementiertes Wizard Modell sowie eine grafische Oberfläche. Sie können entweder ihre eigene grafische Oberfläche verwenden oder das zu dieser Hausaufgabe gehörende Wizard Projekt vom PM Blog herunterladen.
Für die Aufgaben 1-5 ist es ausreichend, wenn die Implementierung für genau 3 Spieler funktionsfähig ist.

Model-View-Controller

In dieser Hausaufgabe soll das Wizard Modell mit der in HA 5 erstellten GUI verbunden werden. Hierzu ist es notwendig eine Reihe von Controllern zu implementieren, die die verschiedenen Modellelemente (z. B. Player, Turn,...) mit ihrer grafischen Darstellung verbinden. In Abbildung 1 ist die bereits aus Übung 7 bekannte MVC Übersicht nochmals dargestellt.

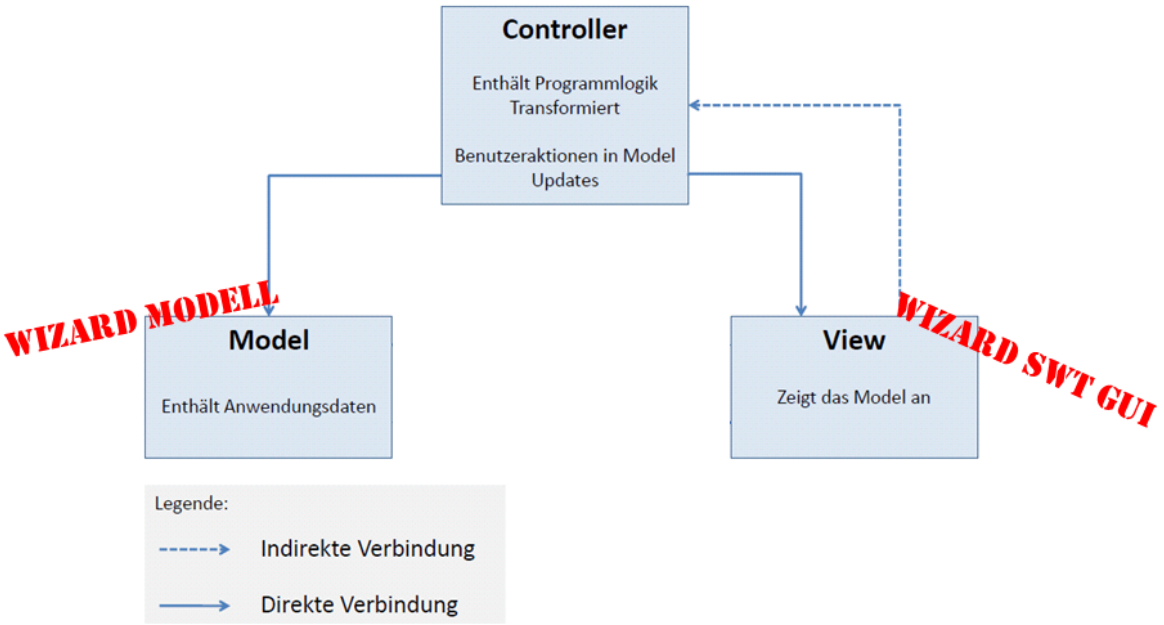

Abbildung 1: Model-View-Controller Entwurfsmuster

StartGameController (5P)

Erstellen Sie eine Klasse `StartGameController`. Der `StartGameController` bekommt im Konstruktur das WizardGame Objekt sowie den in HA 5 entwickelten StartScreen (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Drückt der Nutzer auf „Start Game“, soll zunächst geprüft werden, ob zumindest in die ersten drei Textfelder etwas eingegeben und kein Name doppelt verwendet wurde. Falls nicht, soll ein Popup mit einer entsprechenden Meldung erscheinen(Tip: Hierzu bietet sich die Klasse `MessageDialog` in SWT an).
- Drückt der Nutzer auf „Start Game“ und die Namensvalidierung ist korrekt, soll der GameController instanziiert und gestartet werden. Der StartGameController hat damit seinen Dienst erfüllt und kann gestoppt werden.

- Drückt der Nutzer auf „Quit Game“ soll das Spiel beendet werden.

GameController (7P)

Erstellen Sie eine Klasse `GameController`. Der `GameController` bekommt im Konstruktur das WizardGame Objekt sowie die in HA 5 entwickelte Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Für jeden Spieler instantiiert der `GameController` einen `PlayerController` und startet ihn.
- Die erste Spielrunde soll automatisch gestartet werden.
- Ein `OpenStackController` wird instanziiert und gestartet.
- Der `GameController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente updaten:
  - WizardGame.winner (um das Spielende und den Gewinner zu überwachen)
  - WizardGame.currentTurn (um den Wechsel auf den nächsten Turn zu überwachen)
- Wird ein PropertyChangeEvent empfangen, dass der winner-Link beim WizardGame gesetzt wurde, soll ein Popup mit dem Namen und den Punkten des Gewinners eingeblendet werden. Danach soll das Spiel beendet werden.
- Wird ein PropertyChangeEvent empfangen, dass der currentTurn-Link neu gesetzt wurde, sollte für den neuen Turn ein TurnController instanziiert und gestartet werden.
- Für die Initialisierung der GUI sollte man sich an der Implementierung des mitgelieferten `StartGameControllers` orientieren.

PlayerController (8P)

Erstellen Sie eine Klasse `PlayerController`. Der `PlayerController` bekommt im Konstruktur das Player Objekt sowie die in HA 5 entwickelte Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Der `PlayerController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente updaten:

  - Player.currentPlayerRev (um den aktiven Spieler zu überwachen)
  - Player.forecasts (um Vorhersagen zu überwachen)

- Der `PlayerController` soll das Update für folgende grafische Elemente übernehmen:
  - Die Anzahl der bereits gewonnenen Stiche in dieser Spielrunde
  - Die vorhergesagten Stiche für diese Spielrunde
  - Den Namen des aktiven Spielers anzeigen
  - Die Handkarten des aktiven Spielers einblenden. Dazu können existierende Label Objekte wiederverwendet und nur das anzeigte Bilder verändert werden. Alternativ können Label Objekte bei jeder Änderung auch entfernt und neu erzeugt werden.

    Hinweis: Werden neue Label erzeugt, muss auf dem Elternobjekt die layout()-Methode ausgeführt werden.

- Die Handkarten sollten über das Anklicken ausspielbar sein (siehe `MouseListener` bzw. `MouseAdapter`). Wird eine Handkarte angeklickt, sollen folgende Aktionen ausgelöst werden:
  - Sie wird entsprechend den Wizard-Regeln abgelegt (oder auch nicht ...)
  - Hat jeder Spieler eine Handkarte für diesen Stich abgelegt, wird er ausgewertet
  - Hat nicht jeder Spieler eine Handkarte für diesen Stich abgelegt, ist der nächste Spieler an der Reihe
  - Sind alle Stiche für diese Spielrunde ausgewertet, wird die Spielrunde ausgewertet
  - Ist die Spielrunde ausgewertet, wird automatisch die nächste Spielrunde gestartet

- Am Anfang einer Spielrunde wird von jedem Spieler eine Vorhersage eingeholt. Dies soll über einen Dialog erfolgen, in dem die Vorhersage der Stiche eingegeben wird (siehe `org.eclipse.jface.dialogs.InputDialog`). Es gilt folgende Vereinfachung: Die Handkarten des Spielers müssen nicht angezeigt werden, während er seine Vorhersage eingibt.

OpenStackController (3P)

Erstellen Sie eine Klasse `OpenStackController`. Der `OpenStackController` bekommt im Konstruktur das entsprechende Stack Objekt sowie die in HA 5 entwickelte Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Der `OpenStackController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente updaten:
  - Stack.cards (um das Hinzufügen und Entfernen von Karten zu überwachen)
- Der `OpenStackController` soll das Update für folgende grafische Elemente übernehmen:
  - Anzeige aller Karten auf dem Ablagestapel. Dazu können existierende Label Objekte wiederverwendet und nur das anzeigte Bilder verändert werden. Alternativ können Label Objekte bei jeder Änderung auch entfernt und neu erzeugt werden.

    Hinweis: Werden neue Label erzeugt, muss auf dem Elternobjekt die layout()-Methode ausgeführt werden.

TurnController (4P)

Erstellen Sie eine Klasse `TurnController`. Der `TurnController` bekommt im Konstruktur das entsprechende Turn Objekt sowie die in HA 5 entwickelte Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Der `TurnController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente updaten:
  - Turn.trump (um das Setzen einer Trumpfkarte zu überwachen)
  - WizardGame.currentTurn (um den Wechsel auf den nächsten Turn zu überwachen)
  - Card.color (um das Wechseln der Trumpfkarte zu überwachen) - Die Trumpfkarte ist üblicherweise beim Starten des `TurnController` noch nicht gesetzt, daher erfolgt diese Registrierung bei der Auswertung eines PropertyChangeEvents von Turn.trump
- Der `TurnController` soll das Update für folgende grafische Elemente übernehmen:
  - Die Farbe der aktuellen Trumpfkarte einblenden

  - Die aktuelle Trumpfkarte einblenden

- Ist der Turn des laufenden `TurnController` nicht mehr über den currentTurn Link erreichbar, soll der `TurnController` automatisch gestoppt werden.

Wizard starten (2P)

Erstellen Sie eine Klasse `StartWizard` mit einer `main`-Methode, in der Sie den StartGame-Controller instanziieren und starten. Das Ergebnis beim Ausführen dieser Klasse sollte ein Erscheinen der Login Oberfläche sein.

Zusatzaufgabe 1 - ScoreController (3P)

Erstellen Sie eine Klasse `ScoreController`. Der `ScoreController` bekommt im Konstruktur das WizardGame Objekt sowie die in HA 5 entwickelte Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Der `ScoreController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente updaten:
  - WizardGame.turns (um die Spielrunden zu überwachen)
  - Turn.forecasts (um die Vorhersagen einer Spielrunde zu überwachen) - Die Vorhersagen sind üblicherweise beim Starten des `ScoreController` noch nicht gesetzt, daher erfolgt diese Registrierung bei der Auswertung eines Events von WizardGame.turns.
- Der `ScoreController` soll das Update für folgende grafische Elemente übernehmen:
  - Für jede Spielrunde wird die Spielrundennummer und für jeden Spieler a) die Punkte zum Zeitpunkt dieser Runde und b) seine vorhergesagten Stiche angezeigt.

Zusatzaufgabe 2 - Mehr als 3 Spieler (3P)

Erweitern Sie Ihre Controller aus den Aufgaben 1-5 inklusive Zusatzaufgabe 1, so dass sie mit Spieleranzahlen von 3-6 lauffähig sind.