Die Aufgaben können (müssen aber nicht) in Gruppen von bis zu drei Leuten bearbeitet werden. Falls die Aufgaben in einer Gruppe bearbeitet werden, genügt **eine** Abgabe mit den Namen aller Gruppenmitglieder. Abgabe bis **spätestens Freitag 09.07.2010** per Mail mit dem Betreff **PMSS2010 HA9 <Matrikelnummer>** (z. B. „PMSS2010 HA9 12345678“) an **pm@cs.uni-kassel.de**. Für diese Hausaufgabe gibt es 25 Punkte.

Hinweis 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:

```
PMSS2010 HA9 A<i> <Matrikelnummer>,
```

wobei <i> für die Aufgabennummer steht. Beispiel:

```
PMSS2010 HA9 A1 12345678.
```

Vorbereitung

Zur Bearbeitung dieser Aufgabe benötigen Sie ein fertig implementiertes Mancala Modell sowie eine grafische Oberfläche. Sie können entweder ihre eigene grafische Oberfläche verwenden oder das zu dieser Hausaufgabe gehörende Mancala Projekt vom PM herunterladen.

Model-View-Controller

In dieser Hausaufgabe soll das Mancala Modell mit der in HA 7 erstellten GUI verbunden werden. Hierzu ist es notwendig eine Reihe von Controllern zu implementieren, die die verschiedenen Modellelemente (z. B. Pit, Player, Stone,...) mit ihrer grafischen Darstellung verbinden. In Abbildung 1 ist die bereits aus Übung 9 bekannte MVC Übersicht nochmals dargestellt.

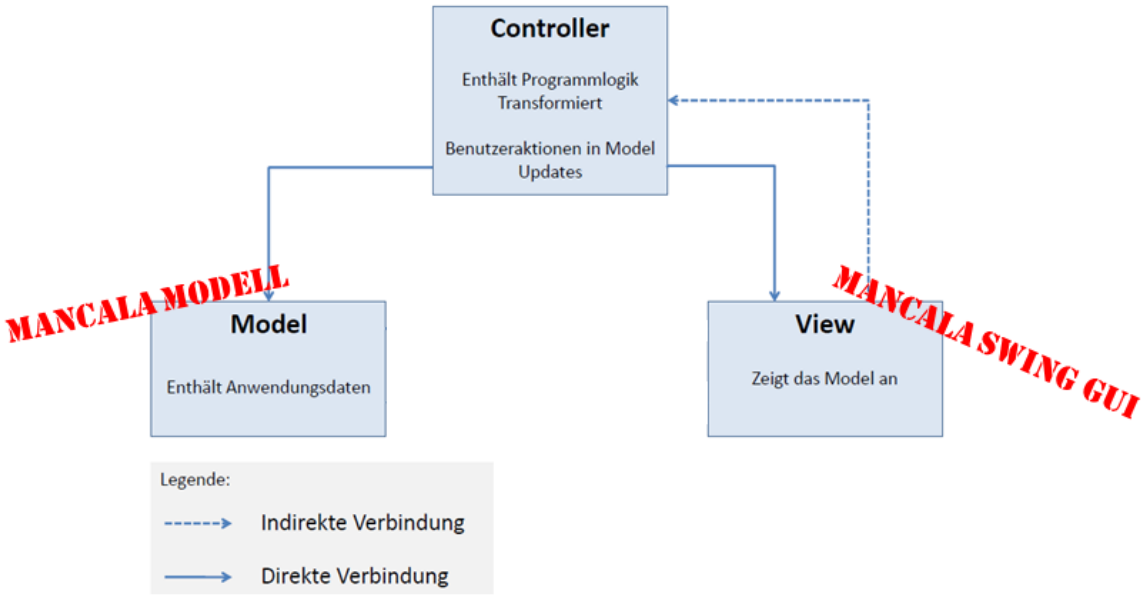

Abbildung 1: Model-View-Controller Entwurfsmuster

LoginController (5P)

Erstellen Sie eine Klasse `LoginController`. Der `LoginController` bekommt im Konstruktur das Mancala Modell Objekt sowie den in HA 7 entwickelten LoginScreen (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Drückt der Nutzer auf „Los“, soll zunächst geprüft werden, ob in die beiden Textfelder etwas eingegeben wurde. Falls nicht, soll Popup erscheinen mit einer entsprechenden Meldung (Tip: Hier können statische Methoden der Klasse `JOptionPane` verwenden werden).
- Drückt der Nutzer auf „Los“ und es wurden zwei Namen eingegeben, soll der `GameController` instanziiert und gestartet werden. Der `LoginController` hat damit seinen Dienst erfüllt und kann gestoppt werden.

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

GameController (6P)

Erstellen Sie eine Klasse `GameController`. Der `GameController` bekommt im Konstruktur das Mancala Modell Objekt sowie die in HA 7 entwickelte Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Für jeden Spieler instantiiert der `GameController` einen `PlayerController` und startet ihn.
- Drückt der Nuter auf „Beenden“ soll zunächst ein Popup mit der Frage erscheinen, ob man wirklich beenden will (Tip: Auch hier kann wieder die Klasse `JOptionPane` verwendet werden). Anwortet der Nutzer mit „Nein“ passiert nichts, antwortet er mit „Ja“ kann das Spiel beendet werden.
- Der `GameController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente upzudaten.
  - Mancala.winner (um das Spielende und den Gewinner zu überwachen)
- Wird ein Event empfangen, dass der Winner Link beim Mancala gesetzt wurde, soll ein Popup mit dem Namen und den Punkten des Gewinners eingeblendet werden. Danach soll das Spiel beendet werden.

PlayerController (8P)

Erstellen Sie eine Klasse `PlayerController`. Der `PlayerController` bekommt im Konstruktur das Player Objekt sowie die in HA 7 entwickelte Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Für jedes Pit welches dem Spieler gehört, soll ein `PitController` instanziiert und gestartet werden.
- Für das Kalah des Spielers soll ein `KalahController` instanziiert und gestartet werden.
- Der `PlayerController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente upzudaten.
  - Player.name (um Namensänderungen zu überwachen)

  - player.kalah (um Anzahl der Steine im Kalah zu überwachen)
  - Mancala.activePlayer (um den aktiven Spieler zu überwachen)
- Der `PlayerController` soll das Update für folgende grafische Elemente übernehmen:
  - Die Schrift des aktiven Spielers rot einfärben.
  - Die Anzahl der Steine im Kalah als Punkte hinter dem Spielernamen.

KalahController (3P)

Erstellen Sie eine Klasse `KalahController`. Der `KalahController` bekommt im Konstruktur das Kalah Objekt sowie den zugehörigen JButton der in HA 7 entwickelten Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Der `KalahController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente upzudaten.
  - Dem im Konstruktor übergebenen Kalah.
- Der `KalahController` soll das Update für folgende grafische Elemente übernehmen:
  - Die Anzahl der Steine im Kalah als Text bei dem im Konstruktor übergebenen JButton setzen.

PitController (3P)

Erstellen Sie eine Klasse `PitController` die von `KalahController erbt`. Der `PitController` bekommt im Konstruktur das Pit Objekt sowie den zugehörigen JButton der in HA 7 entwickelten Spieloberfläche (GUI) übergeben. Der Controller soll folgende Aktionen ausführen:

- Zusätzlich zu den geerbten Aktionen des `KalahController`s soll sich der `PitController` als `ActionListener` an dem im Konstruktor übergebenen Pit (GUI) anmelden.
- Drückt der Nutzer auf den JButton soll auf dem Pit die Methode `moveStones()` aufgerufen werden.

Mancala starten (2P)

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