Die Aufgaben müssen einzeln bearbeitet und abgegeben werden. Die Abgabe muss bis **spätestens Donnerstag 20.12.2012 um 23:59 Uhr** über unser Hausaufgabenabgabesystem **http://seblog.cs.uni-kassel.de/pmws1213/** 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 **25** Punkte.

Hinweise zur Abgabe:

- Die Hausaufgabe als exportiertes Eclipse Projekt (*.zip, **nicht** den gesamten Workspace) abgeben. Das kann mit Hilfe der Eclipse Export Funktion durchgeführt werden. Ist das Projekt nicht korrekt exportiert, kann es bei der Korrektur nicht berücksichtigt werden (es bietet sich also an, den Import des exportierten Projektes auszuprobieren).

  WICHTIG Benennen Sie ihre Projekte für diese und alle zukünftigen Abgaben nach folgendem Schema:

  ```
  PMWS1213_HA<a>_<Matrikelnummer>,
  ```

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

  ```
  PMWS1213_HA7_12345678.
  ```

Vorbereitung

Zur Bearbeitung dieser Aufgabe benötigen Sie ein fertig implementiertes Diaballik Modell sowie eine grafische Oberfläche. Dazu laden Sie das zu dieser Hausaufgabe gehörende Diaballik Projekt vom PM Blog
http://seblog.cs.uni-kassel.de/category/currentterm/pm-ws1213/ herunter und können optional die vorgegebene Oberfläche durch ihre eigene Lösung von HA 6 austauschen.

In dieser Hausaufgabe soll das Diaballik Modell mit der in HA 6 erstellten GUI verbunden werden. Hierzu ist es notwendig eine Reihe von Controllern zu implementieren, die die verschiedenen Modellelemente (z. B. Field, Player, Holder,...) mit ihrer grafischen Darstellung verbinden. In Abbildung 1 ist die bereits aus Übung 7 bekannte MVC Übersicht nochmals dargestellt.

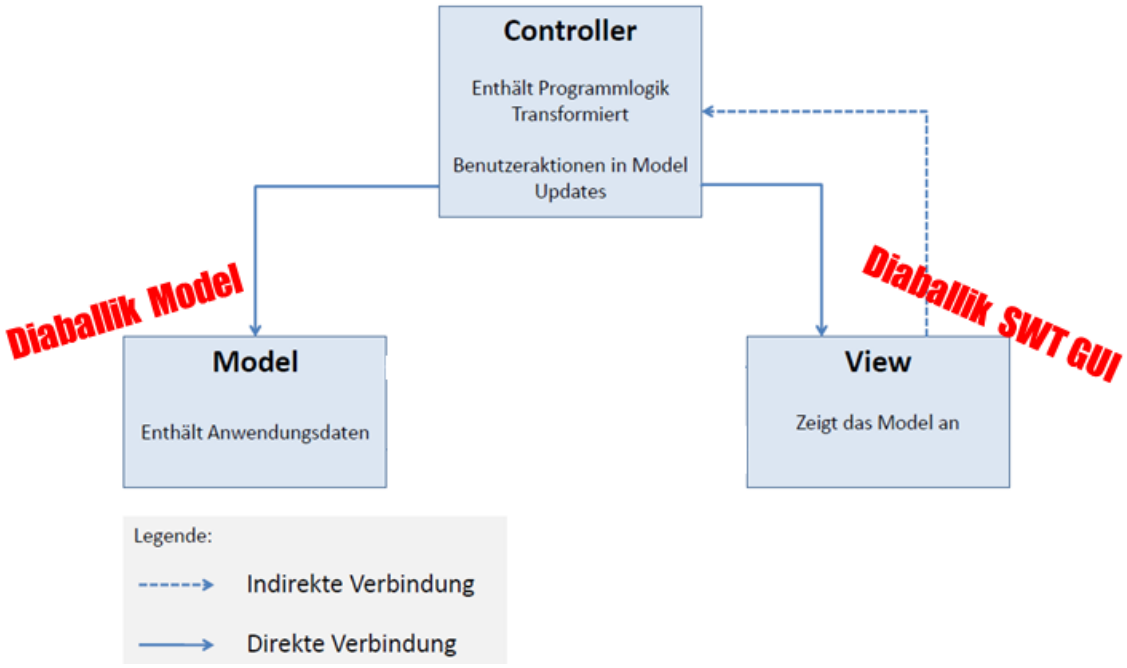

Abbildung 1: Model-View-Controller Entwurfsmuster

Aufgabe 1 - Model-View-Controller (23P)

1.1 - LoginController (5P)

Erstellen Sie eine Klasse `LoginController`. Der `LoginController` bekommt im Konstruktor das DiaballikGame Modell Objekt sowie die in HA 6 entwickelte Loginoberfläche (DiaballikLoginScreen) übergeben. Der Controller soll folgende Aktionen ausführen:

- Drückt der Nutzer auf „Start Game", soll zunächst geprüft werden, ob in beide Textfelder etwas eingegeben wurde und ob die Spielernamen unterschiedlich sind. Falls nicht, soll ein Popup erscheinen mit einer entsprechenden Meldung (Tip: Hier können statische Methoden der Klasse `MessageDialog` verwenden werden).
- Drückt der Nutzer auf „Start Game" und es wurden zwei unterschiedliche Namen eingegeben, wird das Datenmodell durch den Aufruf der init-Methode initialisiert. Nun kann der `LoginController` gestoppt und der `GameController` instanziiert und gestartet werden.
- Drückt der Nutzer auf „Exit Game" soll das Spiel beendet werden.

1.2 - GameController (6P)

Erstellen Sie eine Klasse `GameController`. Der `GameController` bekommt im Konstruktor das DiaballikGame Modell Objekt sowie die in HA 6 entwickelte Spieloberfläche (DiaballikGameScreen) übergeben. Der Controller soll folgende Aktionen ausführen:

- Für jeden Spieler instanziiert der `GameController` einen `PlayerController` und startet ihn.
- Für jedes Spielfeld instanziiert er einen `FieldController` und startet ihn.
- Der `GameController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden Anpassugen der GUI vornehmen:
  - DiaballikGame.currentPlayer (um einen Spielerwechsel zu registrieren)
  - DiaballikGame.winner (um den Gewinner und damit das Spielende zu überwachen)
- Drückt der Nutzer auf „Skip" werden dem aktuellen Spieler seine übrigen Aktionen abgenommen, der nächste Spieler ist an der Reihe und bekommt zwei Move und ein

Pass Objekt. Im Anschluss wird der Skip-Button deaktiviert (button.setEnabled(false)).

- Wird ein Event empfangen, dass der winner-Link beim DiaballikGame gesetzt wurde, soll ein Popup mit dem Namen des Gewinners eingeblendet werden. Nach Klicken des OK-Buttons soll das Spiel beendet werden.

1.3 - FieldController (6P)

Erstellen Sie eine Klasse `FieldController`. Der `FieldController` bekommt im Konstruktor das Field Objekt sowie das zugehörige Label der in HA 6 entwickelten Spieloberfläche (DiaballikGameScreen) übergeben. Der Controller soll folgende Aktionen ausführen:

- Der `FieldController` soll sich bei folgendem Modellelement als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente updaten:
  - Field.holder des übergebenen Field Objekts.
- Der `FieldController` soll das Update für folgende grafische Elemente übernehmen:
  - Wenn eine Holder auf dem Feld steht: Das farblich dem Holder entsprechende Bild in dem übergebenen Label Objekt setzen.
  - Wenn kein Holder auf dem Feld steht: Das dem Feld (leeres Feld, farbiges Startfeld) entsprechende Bild in dem übergebenen Label Objekts setzen.
  - Hier sollen die Methoden der Klasse Util verwendet werden.
- Der `FieldController` meldet sich als `MouseListener` beim im Konstruktor übergebenen Label an.
- Drückt der Nutzer auf das Label sollen folgende Aktionen ausgeführt werden:
  - Wenn noch kein Feld selektiert wurde:
    Prüfen ob auf dem markierten Feld ein Holder des aktiven Spielers steht und in diesem Fall das aktuelle Feld im Game Objekt als selektiert markieren.
    Ansonsten soll der Spieler über ein Popup informiert werden, den Ball oder einen Holder zu selektieren.
    Abschließend soll das aktuelle Feld deaktiviert werden.
  - Wenn bereits ein Feld selektiert wurde:
    Prüfen ob auf dem markierten Feld kein Holder steht und in diesem Fall je nach-

dem ob auf dem vorher selektieren Feld (game.getSelectedField()) ein Holder oder ein Ball liegt die move- bzw. pass-Methode mit dem zuletzt markierten Feld bzw. dem darauf befindlichen Holder aufrufen.
Nachfolgend muss geprüft werden ob jemand gewonnen hat (checkEnd()). Abschließend ist zu prüfen:
Ob der Spieler mindestens eine Aktion ausgeführt hat, wenn ja soll der Skip-Button aktiviert werden.
Oder ob der Spieler noch Aktionen ausführen kann, wenn ja soll der Spieler gewechselt werden.
Nun muss das vorher selektierte Feld wieder aktiviert werden und dem Spiel das vorher selektierte Feld entzogen werden

1.4 - PlayerController (3P)

Erstellen Sie eine Klasse `PlayerController`. Der `PlayerController` bekommt im Konstruktor das Player Objekt sowie die in HA 6 entwickelte Spieloberfläche (DiaballikGameScreen) übergeben. Der Controller soll folgende Aktionen ausführen:

- Für jeden Holder instanziiert der `PlayerController` einen `HolderController` und startet ihn.
  - Der `PlayerController` soll sich bei folgendem Modellelement als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente aktualisieren:
    - Player.actions des übergebenen Player Objekts.
  - Der `PlayerController` soll das Update für folgendes grafische Element übernehmen:
    - Das Label für die Anzahl der verbleibenden Aktionen des übergebenen DiaballikGameScreen Objekts aktualisieren. Es zeigt an, welche Aktionen der aktuelle Spieler noch ausführen kann.

1.5 - HolderController (3P)

Erstellen Sie eine Klasse `HolderController`. Der `HolderController` bekommt im Konstruktor das Holder Objekt sowie die in HA 6 entwickelte Spieloberfläche (DiaballikGameScreen) übergeben. Der Controller soll folgende Aktionen ausführen:

- Der `HolderController` soll sich bei folgenden Modellelementen als `PropertyChangeListener` registrieren und bei Änderungen die entsprechenden GUI Elemente updaten:
  - Holder.ball des übergebenen Holder Objekts.
- Der `HolderController` soll das Update für folgende grafische Elemente übernehmen:
  - Liegt ein Ball auf dem Holder, soll das Holder-Bild des aktuellen Feld-Labels mit einer *decoration* versehen werden.
  - Wird der Ball entfernt, soll wieder das Bild des Holders erscheinen.

Aufgabe 2 - Diaballik starten (2P)

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