Die Aufgaben müssen einzeln bearbeitet und abgegeben werden. Die Abgabe muss **bis spätestens Montag, den 13.05.2013 um 23:59 Uhr** über unser Hausaufgabenabgabesystem **http://seblog.cs.uni-kassel.de/dpss13/** 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.

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 ihr Projekt für diese Abgabe nach folgendem Schema:

  DPSS13_HA<a>_<Matrikelnummer>,

  wobei <a> für die aktuelle Hausaufgabe steht.
  Beispiel:
  DPSS13_HA4_12345678.

Allgemeines
Orientieren Sie sich für die Lösung der Aufgaben an den zugehörigen Übungen und Vorlesungen:
http://seblog.cs.uni-kassel.de/category/currentterm/design-patterns/

Die Benotung beruht auf den Hausaufgaben.

Hierfür werden die gesamten Hausaufgaben minus zwei vom Studenten abgegebenen Hausaufgaben addiert, die mit mehr als 50 % bewertet wurden. Nicht abgegebene Hausaufgaben oder Betrugsversuche bekommen 0 Punkte.

Die Prüfung gilt als nicht bestanden, wenn alle Hausaufgabenpunkte weniger als **50 %** ergeben oder der Student mehr als **zwei Hausaufgaben nicht abgibt**.

Fragen bitte an Stefan.Lindel@cs.uni-kassel.de oder george@uni-kassel.de

Aufgabe 1 (State Pattern) (3P)

- Implementieren Sie das State Pattern am Beispiel eines Bahnübergangs.
- Es werden benötigt:
  - Train
  - Distance (int value = 100)
  - RailroadGate mit einer Methode driveTrainTick(),
    welche distance.getValue() um 10 verringert
    und die Methode driveTrainTick() des aktuellen Zustands aufruft
  - und vier Zustände, die wie in der Vorlesung gezeigt in das Datenmodell eingearbeitet werden sollen:
    - ShowSpeedLimit (Übergang zu BlinkingLight bei distance.getValue() <= 20)
    - BlinkingLight (Übergang zu BarDown bei distance.getValue() <= 10)
    - BarDown (Übergang zu GateOpen bei distance.getValue() < 0)
    - GateOpen (Übergang zu ShowSpeedLimit bei distance.getValue() ==50)
- Das System soll in einem Test, anhand der Durchfahrt eines Zuges, verifiziert werden.

Fragen bitte an Stefan.Lindel@cs.uni-kassel.de oder george@uni-kassel.de

Aufgabe 2 (Model-View-Controller Pattern) (6P)

- Implementieren Sie eine GUI für ein virtuelles Keyboard (Tonleiter: C,D,E,F,G,A,H) welches auf Eingaben hin Töne akustisch abspielt.
- Es soll drei Views, die Eingabe-View, die Keyboard-View und die Senso-View geben.
- Auf der Eingabe-View sollen 7 Buttons mit Beschriftung (C,D,E,F,G,A,H) vorhanden sein. Des Weiteren sollen 7 Tastaturtasten (1-7) jeweils einen unterschiedlichen Ton auslösen.

  Die Eingaben sollen in der Eingabe-View in einem editierbaren Textfeld gesammelt werden. Neben dem Textfeld befindet sich ein Play-Button, der die gesamten gesammelten Töne nacheinander abspielt.
- Auf der Keyboard-View sollen sechs beschriftete Labels (C,D,E,F,G,A,H) zu sehen sein, die auf die jeweiligen Eingaben hin, in der gleichen Farbe (z.B. rot) aufblinken.
  - **Zusätzlich** wäre die View als komplett animiertes Keyboard denkbar.
- Die Senso-View soll ein Tortendiagramm anzeigen, in dem auf die jeweiligen Eingaben hin, für jeden zu implementierenden Ton, das entsprechende Teilstück aufleuchtet.
- Das System soll in einem Test, anhand des Titels „Alle meine Entchen“, verifiziert werden.

Fragen bitte an Stefan.Lindel@cs.uni-kassel.de oder george@uni-kassel.de