SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

UNIKASSEL
VERSITÄT

Design Patterns SS 2015
Hausaufgabe 3

Die Aufgaben müssen von jedem Teilnehmer einzeln bearbeitet und abgegeben werden. Die Abgabe muss **bis spätestens Montag 04.05.2014 um 23:59 Uhr** über unser Hausaufgabenverwaltungssystem **https://se.cs.uni-kassel.de/hms/** erfolgen. Die Abgabe ist nur als einzelne *.zip Datei möglich. Daher müssen alle für eine Abgabe relevanten Daten zu einem solchen Archiv kombiniert werden.

Hinweise zur Abgabe:

- Die Hausaufgabe ist in Form von **drei** exportierten Eclipse Projekten abzugeben. Mit Hilfe der Eclipse Export Funktion ist es möglich drei Projekte **in eine zip-Datei** zu exportieren. Ist ein 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 Abgabe nach folgendem Schema:

  DPSS15_HA<a>_<b>_<Matrikelnummer>,

  wobei <a> für die aktuelle Hausaufgabe und <b> für die Aufgabennummer steht.
  Beispiel für Aufgabe 1:
  DPSS15_HA3_1_12345678.
- Libraries müssen in einen lib oder im Projekt eingefügt werden oder als weitere Projekt mit dem Namen DPSS15_HA<a>_libs_<Matrikelnummer>. Es darf keine Abhängigkeit zu anderen Projekten bestehen oder zu externen Libraries.

Allgemeines:
Orientiert euch für die Lösung der Aufgaben an den zugehörigen Übungen und Vorlesungen:
http://seblog.cs.uni-kassel.de/category/ss15/designpatternss15/

Zusammensetzung der Note:
Es wird n Hausaufgaben geben, von denen **n-2 Sinnvoll** bearbeitet werden müssen.
Sinnvoll bedeutet mindestens 50% der Punkte.
Am Ende ergibt sich die Note aus dem Durchschnitt n-2 besten Abgaben.
Zusätzlich wird es zwei Zusatzaufgaben geben mit denen fehlenden Abgaben ausgeglichen werden können.

Die Zusatzaufgaben werden deutlich umfangreicher als normale Hausaufgaben sein und dienen nicht zur Verbesserung der Note.
Zusatzaufgaben müssen mit **80%** bestanden werden, um wie folgt im Durchschnitt der n-2 besten Hausaufgaben berücksichtigt zu werden:
Wenn bei einer Zusatzaufgabe **100%** der Punkte erreicht wurden, wird sie wie eine **40%** Hausaufgabe bei der Berechnung berücksichtigt. Bei **90%** wird sie wie eine **30%** Aufgabe bewertet, bei **80%** wird sie wie eine **20%** Aufgabe bewertet.

Die Veranstaltung gilt als nicht bestanden, wenn mehr als **zwei Hausaufgaben nicht abgegeben** wurden oder der Mittelwert der zur Benotung herangezogenen Abgaben **weniger als 50%** beträgt.

Fragen bitte an hahn@uni-kassel.de

Aufgabe 1 Visitor Pattern

- Erstellt ein Kassendiagram
- Stellt das Composite Pattern aus Hausaufgabe 2 auf ein Visitor Pattern um
- Erweitert das Datenmodel um Personen
- Implementiert einen Visitor, der nur die Personen zählt, die sich in Fluren befinden (in Etagen aber nicht in Räumen)
- Implementiert einen Visitor, der nur die Personen zählt, deren Namen auf einen übergebenen regulären Ausdruck passt
- Schreibt einen Junit-Test welcher
  - eine sinnvolle Objektstruktur erzeugt wird
  - die Berechnungen durchführt
  - sinnvolle Assertions verwendet werden

Fragen bitte an hahn@uni-kassel.de

Aufgabe 2 Strategy Pattern

Implementiert ein Datenmodell für Bomberman unter Verwendung des Strategy Pattern

- Erstellt ein Kassendiagram
- Bildet dabei folgende Teile des Systems ab:
  - BombermanPlayer
  - BombermanStrategy
- Integriert zwei Attribute vom Typ **`int`** in die Strategy, die über get-Methoden abgefragt werden können:
  - xPosition
  - yPosition
  - numberOfBombs
- Implementiert folgende Strategien inklusive Attribut-Belegungen:
  - moveUp (yPosition--)
  - moveDown (yPosition++)
  - moveLeft (xPosition--)
  - moveRight (xPosition++)
  - blast(numberOfBombs--)
  - stay(numberOfBombs++)
- Schreibt einen Junit-Test in welchem
  - eine sinnvolle Objektstruktur erzeugt wird
  - die Strategien ausgetauscht und abgefragt werden
  - sinnvolle Assertions verwendet werden

Fragen bitte an hahn@uni-kassel.de

Aufgabe 3 Hook Pattern

Erweitert das Strategy Pattern aus Aufgabe 2 so, dass zu jeder Strategy Hooks hinzugefügt werden können

- Erstellt ein Kassendiagram
- Implementiere einen Hook, welcher den Systemzustand ausgibt (in einen globalen String).
- Implementiert folgende Hooks für jede der 6 Strategien aus Aufgabe 2:
  - DebugOutputHook
    - Schreibt in ein Logfile (einen globalen String) welche Strategy gerade aufgerufen wurde
  - RepeaterMonkeyHook
    - Wiederholt die letzte Strategie
  - ForgettingMonkeyHook
    - Macht die letzte Aktion rückgängig
- Schreibt einen Junit-Test in welchem
  - eine sinnvolle Objektstruktur erzeugt wird
  - sinnvolle Assertions verwendet werden