Design Patterns SS 2015
Hausaufgabe 4

Die Aufgaben müssen von jedem Teilnehmer einzeln bearbeitet und abgegeben werden. Die Abgabe muss **bis spätestens Montag 11.05.2014 um 23:59 Uhr** über unser Hausaufgabenverwaltungssystem **https://se.cs.uni-kassel.de/hms/** erfolgen (nicht per Email). 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 **zwei** 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 Chain of Responsibility

Implementiert ein Datenmodell für Bomberman unter Verwendung des Chain of Responsibility Pattern ab.

- Erstellt ein Kassendiagram (SDMlib, xcore, emf, UML Lab, …) (das Datenmodell soll nicht per Hand geschrieben sein)
- Bildet dabei folgende Teile des Systems ab:
  - BombermanPlayer
    - int xPosition, yPosition, numberOfBombs
    - char lastKey
    - void keyPress(char Key)
  - BombermanStrategy
- Implementiert folgende Strategien inklusive Attribut-Belegungen:

  1: moveUp (yPosition--) (Anwendbar, wenn yPosition > 0, lastKey = `W`)

  2: moveDown (yPosition++) (Anwendbar, wenn yPosition < 100, lastKey = `S`)

  3: moveLeft (xPosition--) (Anwendbar, wenn xPosition > 0, lastKey = `A`)

  4: moveRight (xPosition++) (Anwendbar, wenn xPosition < 100 , lastKey = `D`)

  5: blast(numberOfBombs--) (Anwendbar, wenn numberOfBombs > 0 , lastKey = `E`)

  6: stay(numberOfBombs++) (Anwendbar, wenn numberOfBombs < 5)
- Schreibt einen Junit-Test in welchem
  - eine sinnvolle Objektstruktur erzeugt wird
  - die Strategien ausgetauscht und abgefragt werden
  - sinnvolle Assertions verwendet werden

Aufgabe 2 State Pattern

- Implementiert ein State Pattern am Beispiel eines Kaffeevollautomatens
- Erstellt ein Kassendiagram (SDMlib, xcore, emf, UML Lab, …) (das Datenmodell soll nicht per Hand geschrieben sein) für folgende Klassen:
  - CoffeeMachine mit der Methode makeCoffeeStep()
  - BrewGroup (int temp = 23, int coffeePowder = 0, int brewingTime = 0)
  - WaterTank (int fillLevel = 100, int waterTemp = 23)
  - BeanContainer (int fillLevel = 42)
  - CoffeeMill
  - Cup (int fillLevel = 0)
- Die folgenden Zustände sollen wie in der Vorlesung in das Datenmodell eingearbeitet werden:
  - Ready (WaterTank. fillLevel > 23 && BeanContainer.fillLevel >= 5)
    - Aktion: BrewGroup.temp— (bis BrewGroup.temp == WaterTank. waterTemp)
  - BoilWater (BrewGroup.temp <= 90 && WaterTank. fillLevel > 23)
    - Aktion: BrewGroup.temp++
  - MillCoffee (BeanContainer.fillLevel >= 5 && BrewGroup. coffeePowder < 5)
    - Aktionen:
      - BeanContainer.fillLevel—
      - BrewGroup. coffeePowder++
  - BrewCoffee (BrewGroup. coffeePowder == 5 && BrewGroup.temp == 90 && brewingTime < 42)
    - Aktion: brewingTime++
  - FillCup (BrewGroup.brewingTime == 42 && Cup.fillLevel < 23)
    - Aktionen:
      - Cup.fillLevel++
      - WaterTank. fillLevel—
      - BrewGroup.temp --
  - ErrorAddWater (WaterTank. fillLevel < 23)
    - Aktion: WaterTank. fillLevel := 100
  - ErrorAddBeans (BeanContainer. fillLevel < 5)
    - Aktion: BeanContainer. fillLevel := 42

Fragen bitte an hahn@uni-kassel.de

- Schreibt einen Junit Test, der mehrere Kaffee kocht indem er mehrfach CoffeeMachine. makeCoffeeStep aufruft und sinnvolle Assertions verwendet.
- Bei einem Zustandswechsel soll der neue Zustand auf der Konsole ausgegeben werden