SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

UNIKASSEL
VERSITÄT

Design Patterns SS 2015
Hausaufgabe 11

Die Aufgaben müssen von jedem Teilnehmer einzeln bearbeitet und abgegeben werden. Die Abgabe muss **bis spätestens Montag 06.07.2014 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 **einem** exportierten Eclipse Projekten abzugeben. Mit Hilfe der Eclipse Export Funktion ist es möglich mehrere Projekte **in einer 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 Reflection & Annotations

- Erstellt die beiden Annotations @Voodoo und @Magic mit den Attributen int:prio und string:arg
- Schreibt einen MagicVoodooExecuter, welcher den gesamten Classpath durchsucht und
    - Mit der doVoodoo() Methode innerhalb jeder gefundenen Klasse nach parameterlosen und Methoden mit einem String Parameter, die die Annotation @Voodoo verwenden, sucht. Wenn eine Klasse gefunden wird, soll die Methode ausführt werden, falls die Methode statisch ist. Andernfalls soll erst eine Instanz der Klasse erzeugt werden und die Methode dann ausgeführt werden. Bei Methoden mit einem String Parameter soll der Inhalt des Attributes „arg“ der Annotation als Parameter verwendet werden. Das Ganze soll ungeordnet, also in der Reihenfolge, wie die Klassen gefunden worden sind, ausgeführt werden.
    - Die doVoodooOrdert() soll wie doVoodoo() funktionieren, jedoch vor dem ausführen der Methode, die Ausführreihenfolge nach dem „prio“ Attribut der Annotation ordnen
    - Mit der doMagic() Methode soll bei jeder Klasse geprüft werden, ob die @Magic Annotation verwendet wird. Also in diesem Fall bei der Klasse. Wenn das so ist, soll eine Instanz der Klasse erzeugt werden, und die toString() Methode des Objektes aufgerufen werden.
      Bei Klassen, die einen Konstruktor haben, der genau einen String Parameter erwartet, verwendet das „arg“-Attribut der @Magic Annotation
    - Die doOrdertVoodooMagic() soll die Konzepte der vorigen Methoden kombinieren ;) (gibt Zusatzpunkte)
- Schreibt mehrere Klassen, welche die Annotations verwenden und erstellt einen Sinnvollen JUnit Test