Die Aufgaben müssen von jedem Teilnehmer einzeln bearbeitet und abgegeben werden. Die Abgabe muss **bis spätestens Montag 16.06.2014 um 23:59 Uhr** über unser Hausaufgabenverwaltungssystem **https://se.cs.uni-kassel.de/hms/** 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 ist in Form von **zwei** exportierten Eclipse Projekten abzugeben. Mit Hilfe der Eclipse Export Funktion ist es möglich mehrere 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:

  DPSS14_HA<a>_<b>_<Matrikelnummer>,

  wobei <a> für die aktuelle Hausaufgabe und <b> für die Aufgabennummer steht.
  Beispiel für Aufgabe 1:
  DPSS14_HA8_1_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-patterns2014/

Die Benotung ergibt sich aus den Hausaufgaben, wobei eine Abgabe ausgelassen werden darf. Für die Note wird die nicht abgegebene Abgabe, beziehungsweise die Abgabe mit der geringsten Prozentzahl, nicht beachtet. Die Endnote ergibt sich aus dem Mittelwert der erreichten Prozentpunkte der übrigen Abgaben.

Die Veranstaltung gilt als nicht bestanden, wenn mehr als **eine Hausaufgabe nicht abgegeben** wurde oder der Mittelwert der zur Benotung herangezogenen Abgaben **weniger als 50%** beträgt.

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

Aufgabe 1 Singleton Pattern

Implementieren Sie ein Datenmodell für ein Parallel-Worker-System unter Verwendung des Singleton Pattern.

- Bilden Sie dabei folgende Teile des Systems ab:
  - ProducerThread
  - Worker
  - Task mit Attribute: value::int
- Die Producer-Threads schreiben eine Aufgabe in eine Liste der Worker-Instance. Die Worker-Instance entnimmt die Aufgaben aus der Liste und schreibt sie in eine lokale Datei (Persistierung über JSON).
- Schreiben Sie einen Junit-Test in welchem
  - eine sinnvolle Objektstruktur erzeugt wird
  - **5** ProducerTheads in einer Schleife jeweils zufällig **1-5** Aufgaben erstellen, welche sie an den Worker zurückgeben
  - sinnvolle Assertions verwendet werden

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

Aufgabe 2 Facade Pattern

Implementieren Sie ein Datenmodell für ein PC-System unter Verwendung des Facade-Pattern.

- Bilden Sie dabei folgende Teile des Systems ab:
  - PC
  - USB-Device
  - mindestens einen anderen Treiber nach Wahl
  - Driver (Interface)
  - und die für das Pattern notwendigen Klassen
- Die Treiber müssen folgende Schnittstelle unterstützen
  - getCaption() :: String
  - getVersion() :: String
  - init(Driver) :: boolean
  - getValue() :: Object
  - addListener(PropertyChangeListener) :: boolean
- Es soll ein USB-Device erstellt werden, welches ein boolean Attribut `mount` besitzt. Der PC soll auf die Veränderung des Attributs reagieren und die getMounted() :: String soll „USB is mounted“ zurückliefern.
- Schreiben Sie einen Junit-Test in welchem
  - eine sinnvolle Objektstruktur erzeugt wird
  - getMounted() und die eigenen getXYZ()-Methoden aufgerufen werden
  - sinnvolle Assertions verwendet werden

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