Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Übung 9 Version: 1.0 14. 12. 2009

---

1 Mini Quiz

Objekte und Klassen

- ☐ Java und Scheme sind objektorientierte Sprache.
- ☐ Ein Objekt kann in Java Attribute und Methoden enthalten.
- ☐ Ein Objekt ist eine Instanz einer Klasse.
- ☐ Eine Klasse instanziiert für gewöhnlich ein Objekt.

Compiler vs. Interpreter

- ☐ Sowohl Scheme als auch Java sind Maschinensprachen.
- ☐ Java nutzt eine Virtual Machine (VM).
- ☐ Java-Programme sind nur lauffähig auf dem Maschinentyp, auf dem sie kompiliert wurden.
- ☐ Byte-Code Programme sind schneller als Maschinenprogramme.
- ☐ Ein Interpreter führt ein Programm einer bestimmten Programmiersprache direkt aus.

Packages

- ☐ Packages sollen Klassen bündeln, die im Hinblick auf Zuständigkeit zusammengehören.
- ☐ Mit einer "Wildcard" (*) beim Import kann man auf alle Unterklassen eines Packages zugreifen.
- ☐ Eine Klasse kann in Java mehreren Packages angehören.

2 Fragen

1. Erklären Sie die folgenden Begriffe in eigenen Worten: **Klasse**, **Objekt** und **Instanz**. Geben Sie ein Beispiel für jeden Begriff an.
2. Was ist ein Konstruktor? Wofür wird dieser benötigt? Wie sieht ein minimaler Konstruktor aus?
3. Nennen Sie die wichtigsten Methoden der *Assert*-Klasse, die man bei einem typischen JUnit Testcase benötigt.
4. Nennen und beschreiben Sie die vier Phasen der Übersetzung (Kompilierung) eines Programms.

3 Modellierung und Test eines virtuellen Autos

1. Schreiben Sie eine Klasse `Car` zur Repräsentation von Autos, die folgende Anforderungen erfüllen soll:

   - Ein Auto hat einen Namen vom Typ *String* und einen Kilometerstand (*mileage*) vom Typ *double*. Beide Attribute sollten *private* sein.
   - Der Konstruktor soll einen String als Parameter erhalten, der den Namen des Autos angibt. Der Konstruktor soll den Namen des Autos setzen und den Kilometerstand auf 0.0 setzen.
   - Schreiben Sie die Getter-Methoden **public double** getMileage() und **public** String getName(), um auf die Attribute der Klasse `Car` zuzugreifen.
   - Schreiben Sie die Methode **void** drive(**double** distance), die eine Distanz in Kilometern als Argument erhält und den Kilometerstand entsprechend aktualisiert.
   - Vergessen Sie nicht die Verwendung von JavaDoc-Kommentaren für alle Elemente, die nach außen sichtbar sind!

   Hinweis: Eclipse kann Ihnen viel von der Arbeit abnehmen, wenn Sie die Menüeinträge im Menü "Source" etwas erkunden. Allerdings sollten Sie auch in der Lage sein, diese Aufgabe *ohne* Eclipse zu bearbeiten (Stichwort: Programmierung auf Papier in der Klausur)!

2. Schreiben Sie nun einen minimalen JUnit-Testcase, der die beiden folgenden Tests abdecken soll:

   - Anlegen mindestens eines "Test-Autos" in einer mit @Before annotierten Methode. Die anderen Tests können dann direkt auf diesem Auto arbeiten. Denken Sie daran, dass diese Methode vor *jeder* Testmethode aufgerufen wird.
   - Nun testen Sie bitte den Konstruktor (der Erzeugung des Objekts) und die Methode String getName(). Dazu soll nach dem Anlegen des Objektes dessen Name einmal mit dem tatsächlichen und einem mit einem falschen Namen überprüft werden sowie geprüft werden, ob der Anfangskilometerstand 0.0 beträgt. Benutzen Sie hier bitte sowohl *assertFalse(String message, boolean condition)* als auch *assertEquals(String expected, String actual)* und übergeben Sie für *assertFalse* einen passenden String als (mögliche) Fehlermeldung.
   - Testen Sie die Methoden **void** drive(**double** distance) und **double** getMileage(). Dazu soll das Auto erst 74.3 km "fahren", der Kilometerstand überprüft werden, dann soll das Auto weitere 26.8 km "fahren" und der Kilometerstand erneut überprüft werden.

4 Objekte und deren Analyse

Bitte lesen Sie zunächst die gegebene Java-Klasse und beantworten Sie anschließend die einzelnen Fragen.

```
1  public class P {
2
3    private Long method1(Long x, Long y) {
4      if (y == 1)
5        return x;
6      return x + method1(x, y - 1);
7    }
8
9    private Long method2(Long x, Long y, Long z) {
10     z = y - 1;
11     if (y == 1)
12       return x;
```

```
13      return method1(x, method2(x, y - 1, z));
14    }
15  }
```

1. Wieviele primitive Datentypen werden in der Klasse `P` verwendet?
2. Was würde folgender Aufruf innerhalb der (in der Klasse P nicht angegebenen) *main-Methode* als Ergebnis liefern?

```
1  /**
2   * Run the program using method2...
3   * @param args command-line arguments (ignored).
4   */
5  public static void main(String[] args) {
6    P p = new P();
7
8    Long a = new Long(2),
9         b = new Long(5),
10        c = new Long(a - b);
11
12   System.out.print("Result: " + p.method2(a, b ,c));
13 }
```

3. Berechnen Sie auch das Ergebnis für a=4, b=3 und c=2.
4. Was für einen Zweck erfüllt der Algorithmus, der sich aus method1(..) und method2(..) zusammensetzt? Beachten Sie die verschachtelte Rekursion!

   Hinweis: Die Klasse `Math`, die Sie kennen sollten, enthält diesen Algorithmus (wenn auch in einer leicht abgewandelter Form).
5. In der oberen Klasse gibt es eine überflüssige Variable. Ermitteln Sie diese und erklären Sie, warum diese nicht benötigt wird.

5 Mehr JUnit...

In dieser Aufgabe wollen wir ein paar fortgeschrittene Techniken zum Testen mit JUnit betrachen.

1. Gegeben sei folgende Klasse *Random* mit (bewusst) unvollständiger Kommentierung:

```
1  /**
2   * Random class for GdI / ICS exercise sheet 9.
3   *
4   * @author Oren Avni / Guido Roessling
5   * @version 1.0
6   */
7  public class Random {
8
9    /**
10    *
11    *
12    * @param m
13    * @param s
14    * @return an int array that does something...
15    */
16   public int[] doSomething(int m, int s) {
17     int i = 0;
18     int[] arr = new int[s];
19
20     while (i < arr.length) {
21       arr[i] = (int) (m * (Math.random())) + 1;
```

```
22          i++;
23        }
24
25        return arr;
26      }
27    }
```

a) Erklären Sie zunächst, was für einen Zweck die Methode *doSomething* verfolgt.

Hinweis: `Math.random()` liefert einen zufälligen Wert $x \in [0, 1[$ zurück. (**int**)value wandelt eine Gleitkommazahl in die entsprechende Ganzzahl; (**int**)5.722 ist also die Ganzzahl 5.

b) Schreiben Sie eine JUnit-Testklasse, die ein privates Attribut vom Typ *Random* anlegt und anschließend für die Ergebnisse des Aufrufes *doSomething(50, 100)* prüft, ob die einzelnen Werte des Arrays die gewünschten Eigenschaften haben. Verwenden Sie für den Test *assertTrue(String messageOnFail, boolean condition)*.

c) Schreiben Sie nun einen JUnit-Testcase, der genau wie der vorherige Test arbeitet, nur für den Aufruf *doSomething(100, 2000000)*. Bitte beachten Sie, dass der Test **scheitern soll**, falls die Ausführung länger als 50 Millisekunden dauert. Wenn Ihr Rechner nicht sehr schnell ist, sollte dieser Test scheitern!

2. Wir betrachten nun die folgende Klasse *Strange* sowie den entsprechenden JUnit-Testcase:

```
1  public class Strange {
2    public int getAvg(int[] array) {
3      int temp = 0;
4
5      for (int i = 0; i <= array.length; i++) {
6        temp += array[i];
7      }
8
9      return (temp / array.length);
10   }
11 }
```

```
1  import static org.junit.Assert.assertEquals;
2  import org.junit.Test;
3
4  public class StrangeTest {
5    @Test
6    public void testStrange() {
7      // create Strange instance
8      Strange strange = new Strange();
9      // check value
10     assertEquals(5, strange.getAvg(new int[] { 1, 3, 5, 7, 9 }));
11   }
12 }
```

a) Was passiert, wenn Sie den Test ausführen? Was müssen Sie an der Klasse *Strange* ändern, damit der Testcase durchläuft und keine Fehlermeldungen ausgibt?

Hinweis: Es reicht eine einzige Änderung.

b) Die Methode *getAvg(int[] array)* besitzt noch einen *Intentionsfehler*. Finden Sie ihn? Wieso läuft der Testcase trotzdem fehlerfrei, nachdem Sie die erste Teilaufgabe gemeistert haben?

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren. Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Portal auch für sie bewertet werden kann.

Abgabedatum: Freitag, 15. 01. 2010, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß der Vorgaben zu kommentieren (Scheme: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Java: JavaDoc). Zerlegen Sie Ihren Code sinnvoll und versuchen Sie, wo es möglich ist, bestehende Funktionen wiederzuverwenden. Wählen Sie sinnvolle Namen für Hilfsfunktionen und Parameter.

6 Objekt-Orientierte Modellierung (OOM) (10 Punkte)

In dieser Aufgabe sollen Sie mit Hilfe von Vererbungshierarchien und abstrakten Klassen eine Erweiterung des Kalendars von Übungsblatt 8 modellieren. Hierbei geht es darum, weitere Klassen von eintragbaren Objekten zu implementieren und ein einfaches Benutzerkonzept umzusetzen. Ein Vorlage zum Import in Eclipse finden Sie im Portal.
Es gibt bei OOM nicht unbedingt richtig und falsch, alles ist potentiell richtig, wenn es denn die Anforderungen erfüllt. Mit der Zeit haben sich allerdings Musterlösungen für bestimmte Problemstellungen herauskristallisiert. Diese sogenannten *Design Pattern*[1] haben feststehende Begriffe für die Modellierung von Softwaresystemen etabliert. Für die vorliegende Aufgabe ist Wissen um die verschiedenen Pattern nicht notwendig, schadet aber auch nicht.
Da Sie sich in dieser Aufgabe mit OOM auseinandersetzen sollen, werden wir nur das zu erstellende System beschreiben (mit Einschränkungen, um das Korrigieren handhabbar zu halten). *Wie* Sie dieses modellieren, ist Ihnen überlassen, sofern es die gegebenen Anforderungen erfüllt.
Allerdings sollten Sie einige grobe Richtlinie für gutes OOM einhalten. So sollte das Duplizieren von Quellcode tunlichst vermieden werden, da Sie sonst Änderungen und Korrekturen an mehreren Stellen tätigen müssen, was schnell unübersichtlich wird. Daneben sollten Sie ein Auge auf die Wiederverwendbarkeit und Erweiterbarkeit haben, was in unserem Fall aber noch nicht wirklich kritisch ist. Außerdem sollte der Zugriff auf Attribute von außerhalb über `getter` und `setter` Methoden erfolgen und der direkte Zugriff möglichst restriktiv gehandhabt freigegeben werden (Stichwort **private**, **protected**, **public**).

1. Im neuen Kalendar soll es fünf verschiedene Typen von Einträgen geben können:

   a) *Einfache Notizen* als Objekte der Klasse `Note`. Diese haben zusätzlich zur Beschreibung von `CalendarEntry` im Allgemeinen noch ein zusätzliches Attribut: einen String namens `details`.

   b) *Erinnerungen* als Objekte der Klasse `Reminder`. Neben den herkömmlichen Attributen eines Kalendareintrags sollen diese einen weiteren Zeitpunkt (`alarmTime`) enthalten, an dem ein Benutzer an den Eintrag erinnert werden soll. Dieser muss vor dem eigentlichen Zeitpunkt liegen.

   c) Geplante *Zusammentreffen* von Entwicklern sollen als `Meeting` zusammen mit ihrem jeweiligen Zeitraum und den Teilnehmern (`participants`) eingetragen werden können.

   d) Um die Krankheit und damit verbundene Abwesenheit eines Benutzers für einen bestimmten Zeitraum anzuzeigen, sollen Objekte der Klasse `Illness` in den Kalendar eintragbar sein.

[1] `http://de.wikipedia.org/wiki/Design_Pattern`

e) *Urlaub* von Benutzern soll als Objekt der Klasse `Vacation` mit dem Datum der Rückkehr in den Kalendar eingetragen werden können.

2. Daneben soll es auch zwei verschiedene Klassen von Benutzern geben: *Entwickler* als Objekte der Klasse `Developer` sowie *Verwaltungsangestellte* als Objekte der Klasse `Secretary`. Beide sollen jeweils Felder für den Vor- und Nachnamen enthalten (`givenName` und `familyName`).

3. Neben der Bereitstellung oben genannter Entitäten gibt es noch verschiedene Anforderungen, die der Kalendar und die Eintragungen erfüllen müssen.

   a) In Einträgen für Krankheit und Urlaub soll optional ein Stellvertreter als `delegate` angegeben werden können. Für diesen sind die Einträge auch per `listEntries` sichtbar, wenn sie als privat markiert wurden (1 Punkt).

   b) Ähnlich wie ein Kalendar mehrere Einträge hat, soll ein Zusammentreffen eine Liste seiner Teilnehmer als `participants` enthalten. Ausschließlich Entwickler können an diesen Treffen teilnehmen (1 Punkt).

   c) Alle Einträge, welche eine Zeitdauer beschreiben (`Vacation`, `Illness`, `Meeting`), sollen eine Funktion `public int getDuration(int field)` bereitstellen, die als Argument eine der folgenden Konstanten der Klasse `java.util.Calendar` entgegennimmt:

      - `Calendar.DATE`
      - `Calendar.HOUR`
      - `Calendar.MINUTE`
      - `Calendar.SECOND`

      Die vordefinierten Konstanten können benutzt werden, um eindeutig das gemeinte Feld eines Datums in der Klasse *GregorianCalendar* zu beschreiben. Entsprechend soll diese Methode die Dauer in Tagen, Minuten, Stunden oder Sekunden zurückliefern, jeweils abgerundet auf die nächste ganze Zahl (2 Punkte).

      Hinweis: Die Konstanten werden in Java in der Klasse `java.util.Calendar` definiert. Da wir eine eigene `Calendar` Klasse bereitstellen, würde es hier zu Namenskonflikte kommen. Eine einfache Möglichkeit, diese aufzulösen ist, die gleichen Felder aus der den `java.util.Calendar` beerbenden Klasse `java.util.GregorianCalendar` zu verwenden. Auf diese Weise müssen Sie nicht den vollqualifizierten Namen der `Calendar` Klasse verwenden.

   d) Ferner sollen alle Einträge, welche eine Dauer haben, eine Methode `public boolean spans(GregorianCalendar date)` bereitstellen, welche für ein übergebenes Datum als `GregorianCalendar` überprüft, ob dieses Datum innerhalb des Zeitraumes liegt (1 Punkt).

   e) Alle Einträge mit Dauer sollen darüberhinaus eine Methode `public String toString()` bereitstellen, welche an die bekannte Ausgabe von `CalendarEntry` noch die Dauer des größten Datumsfeldes != 0 aus `getDuration()` als Richtwert anhängt (2 Punkte):

      ```
      .... - about 3 day(s)
      .... - about 1 hour(s)
      ```

      Hinweis: Mit dem Prefix `super.` können Sie innerhalb einer Java Klasse explizit auf die Attribute und Methoden der Basisklasse zugreifen, ähnlich wie Sie per `this.` explizit auf die Attribute und Methoden der aktuellen Klasse zugegriffen haben.

   f) Die Klasse `Calendar` soll um eine Methode

      public Calendar getBetween(GregorianCalendar start , GregorianCalendar end)

      erweitert werden, welche zwei Datumsangaben `start` und `end` als GregorianCalendar konsumiert und einen neuen Kalendar erzeugt, der nur Einträge enthält, deren Beginn zwischen diesen beiden Datumsangaben liegt (1 Punkt).

Hinweis: Um ein zeitnahes und konsistentes Korrigieren zu ermöglichen, sind die Klassen der möglichen Kalendareinträge sowie die der Benutzer mit ihren Konstruktoren vorgegeben. Definieren Sie *keine weiteren* Konstruktoren, sondern stellen sie Zugriffsmethoden für die zusätzlichen Attribute bereit. Nutzen Sie dabei die Namenskonventionen für `setter` und `getter` aus Eclipse, wie Sie auch in der letzten Übung verwendet wurden.

4. Wenn ein Objekt eine Menge von anderen Objekten als Liste enthält, sollen Sie zusätzlich zu den `setter` und `getter` Methoden für die Listen als solche Listen auch Methoden `addFieldName`, `removeFieldName` und `hasFieldName` zur Verfügung stellen; der Feldname ist dabei jeweils im Singular anzugeben. Als Beispiel für das Attribut `calendarEntries` würden fünf Methoden zu implementieren sein.

   ```
   -> boolean hasCalendarEntry(CalendarEntry entry)
   -> void addCalendarEntry(CalendarEntry entry)
   -> void removeCalendarEntry(CalendarEntry entry)
   -> CalendarEntry[] getCalendarEntries()
   -> void setCalendarEntries(CalendarEntry[] entries)
   ```

   Alle diese Methoden erwarten ein Objekt von dem zur Liste passenden Typ und geben jeweils den Typ `void`, `void` und `boolean` zurück. Verhindern Sie dabei das mehrfache Hinzufügen des selben Objektes in eine Liste (3 Punkte).

   Hinweis: Das gilt natürlich auch für die bestehende Klasse `Calendar`.