Universität des Saarlandes
Lehrstuhl für Softwaretechnik
Prof. Andreas Zeller
Valentin Dallmeier, Silvia Breu, Stephan Neuhaus

9. Übung Softwaretechnik

Software-Design und Aspektorientierte Programmierung

Abgabe: 13.01.06, 12 Uhr

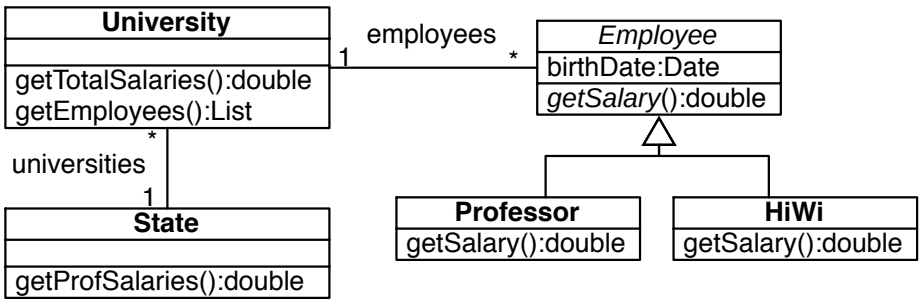
1 Design (5 Punkte)

In der Vorlesung haben sie die 4 Grundprinzipien des objektorientierten Modells *Abstraktion*, *Einkapselung*, *Modularität* und *Hierarchie* kennengelernt. Geben Sie für jeden Eintrag in der untenstehenden Tabelle an, ob das jeweilige aus objektorientierten Programmiersprachen bekannte Konzept zu dem angegebenen Prinzip beiträgt oder nicht und begründen Sie jeweils kurz Ihre Antwort.

	Abstraktion	Einkapselung	Modularität	Hierarchie
Klassen				
Schnittstellen				
Sichtbarkeitsmodifizierer				
Vererbung				
Dynamische Bindung				

2 Design-Prinzipien (10 Punkte)

Eine mögliche Realisierung einer Universitätsverwaltung könnte wie in diesem UML-Diagramm aussehen: Die



Universitäten werden vom Staat verwaltet. Eine Instanz von `University` verwaltet eine Liste von Angestellten (`employees`) der Universität. Unterschiedliche Arten von Angestellten werden als konkrete Subklassen der abstrakten Oberklasse `Employee` modelliert. Diese definiert eine abstrakte Methode `getSalary()`, die das Gehalt des Angestellten zurückgibt.

a, Die Implementierung von `getTotalSalaries()` in der Klasse `University` sieht so aus:

```
  public double getTotalSalaries() {
    double sum=0;
    Iterator i = employees.iterator();
    while (i.hasNext()) {
      sum+=((Employee) i.next()).getSalary();
    }
    return sum;
  }
}
```

Beschreiben Sie mit eigenen Worten das Open-Close- und das Abhängigkeits-Prinzip und erläutern Sie deren Umsetzung anhand des angegebenen Beispiels.

b, Der Staat möchte gerne das Gehalt aller angestellten Professoren ermitteln. Die Implementierung dieser Methode in der Klasse `State` sieht so aus:

```
public double getProfSalaries() {
  University u; Employee e; int sum = 0;
  for (int i=0; i < universities.size(); i++) {
    u = (University) universities.get(i);
    for (int j=0; j < u.getEmployees().size(); j++) {
      e = (Employee) u.getEmployees().get(j);
      if (e instanceof Professor) {
        sum += e.getSalary();
      }
    }
  }
  return sum;
}
```

Welche Probleme hat diese Implementierung? Ändern Sie die Implementierung so, dass sie den in der Vorlesung vorgestellten Prinzipien genügt. Bei Bedarf können Sie auch den Entwurf ändern.

3 Liskov-Prinzip (5 Punkte)

In der Vorlesung haben Sie das Liskov-Prinzip kennengelernt. Es verlangt für überschriebene Methoden in abgeleiteten Klassen *Verhaltenskonformanz*, d.h. eine gleichstarke oder schwächere Vorbedingung und eine gleichstarke oder stärkere Nachbedingung. In dieser Aufgabe sollen Sie für ein gegebenes Beispiel überprüfen, ob das Liskov-Prinzip erfüllt ist:

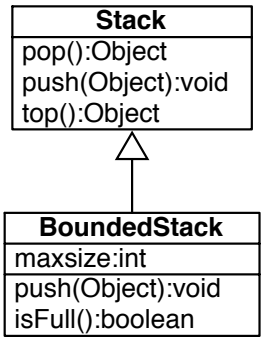
`Stack` ist eine Implementierung eines Stacks der beliebig gross werden kann. Die Implementierung bietet die üblichen Methoden `push()`,`pop()` und `top()`. `BoundedStack` ist eine Unterklasse von `Stack` und kann nur bis zu einer gewissen Grösse `maxSize` wachsen. Sie überschreibt `push()` aus der Oberklasse und bietet eine zusätzliche Methode `isFull()`, welche prüft, ob der Stack die maximale Grösse erreicht hat.

a, Geben Sie zunächst die Vor- und Nachbedingungen für die Methode `push` in `Stack` an.

b, Geben Sie an, wie sich `push` in `BoundedStack` verhalten soll, wenn durch den Aufruf die maximale Grösse des Stacks überschritten wird.

c, Geben Sie die Vor- und Nachbedingungen für die verfeinerte Methode `push` in `BoundedStack` an.

d, Ist `BoundedStack` eine Erweiterung von `Stack` nach dem Liskov-Prinzip oder nicht? Begründen Sie Ihre Antwort.



4 Aspektorientierte Programmierung (10 Punkte)

Machen sie sich anhand der Vorlesungsfolien und der Quickreference aus dem *Aspectj Programmer's Guide* mit den Ansatzpunkten (Join Points, PointCuts, Advice, Aspect) aspektorientierter Programmierung vertraut.

a, Schreiben Sie einen Aspekt, der alle Aufrufe von Funktionen aus dem `java.util`-Paket auf der Konsole protokolliert.

b, Schreiben Sie einen Aspekt, der `java.util.Vector` um eine Funktion `countNullItems()` erweitert. Diese Funktion soll zählen, wie oft der Vector `null` als Eintrag enthält und das Ergebnis zurückgeben.

c, Seit JDK 1.3 verfügt Java über eine neue Klasse `java.lang.StrictMath`, die striktere Varianten zur Berechnung mathematischer Funktionen bereithält als die ursprüngliche Implementierung in `java.lang.Math`. Schreiben Sie einen Aspekt, der für jeden Aufruf von `java.lang.math.sqrt(double a)` das Ergebnis mit dem von `java.lang.StrictMath.sqrt(a)` vergleicht und eine eventuelle Abweichung ausgibt.

Alle Ihre Aspekte müssen mit AspectJ-Version 1.5 kompilieren. Die Homepage von AspectJ finden Sie unter `http://www.aspectj.org`. Die Java Virtual Machine können Sie unter `http://java.sun.com` herunterladen. Geben Sie einen Ausdruck der Aspekte ab. Schicken Sie zusaetzlich *eine* Mail mit dem Betreff *Exercise 9* an `se0506@st.cs.uni-sb.de` und hängen Sie jeden Aspekt in einer eigenen Datei an diese Mail an. Vergessen Sie nicht, die Matrikelnummern der abgebenden Gruppe in der Mail anzugeben.

Hinweis: Falls Sie bei Aufgabe b, oder c, Schwierigkeiten haben, mit ihren Aspekten `java.util.Vector` und `java.lang.Math` zu verändern, dürfen Sie alternativ auch die Klassen `MyVector` und `MyMath` erweitern. Diese Klassen können Sie auf der Homepage der Übung `http://www.st.cs.uni-sb.de/edu/se/2005/uebung.php` herunterladen.

Abgabe

Bilden Sie Teams aus je zwei Studenten, erarbeiten Sie die Lösung *gemeinsam* und reichen Sie *eine Lösung pro Team* ein. Drucken Sie Ihre Lösungen aus, klammern Sie sie zusammen und werfen Sie sie in die mit "Softwaretechnik" beschrifteten Übungskästen vor Hörsaal 1 in Gebäude E1 1 (45). Vergessen Sie nicht, Ihre Lösungen für Aufgabe 4 wie beschrieben zusätzlich zum Ausdruck auch per Mail abzugeben.

Fragen?

Fragen zu diesem Übungsblatt können sie in Ihrer Übungsgruppe stellen.