HUMBOLDT-UNIVERSITÄT ZU BERLIN
INSTITUT FÜR INFORMATIK

Sommersemester 2010

Prof. Dr. Holger Schlingloff
Dr. Anatolij Zubow

Übungen zu Grundlagen der Programmierung

Übungsblatt Nr. 6 (30 Punkte)

Ausgabedatum: 28. 6. 2010, 17:00 – Abgabedatum: 12. 7. 2010, 8:00

Die Lösung der Aufgaben soll in Gruppen zu je zwei Studierenden erfolgen. Die Aufgaben sind elektronisch abzugeben. Auf jedem Lösungsblatt sind Name und Matrikelnummer der beiden Gruppenmitglieder anzugeben! Von jedem Blatt werden mindestens 20% der Punkte (also 6) verlangt.

Aufgabe 1 (8 Punkte)

Gegeben sei die folgende Klasse Quadrat:

```
class Quadrat {

    protected int len; // Seitenlänge

    public Quadrat(int len) {
        this.len = len;
    }

    public int getArea() {
        return len * len;
    }
}
```

a) 4 Punkte

Um die Gleichheit zweier *Quadrat*-Instanzen zu überprüfen, verwendet man die Methode *equals()*. Überschreiben sie diese Methode (*equals()* ist bereits in der Klasse *Object* definiert).

Hinweis: Die Lösung ist kein Zweizeiler!!!

b) 3 Punkte

Erweitern Sie die Klasse so, dass eine neue Klasse *Rechteck* entsteht (besitzt zusätzlich Seitenhöhe). Schreiben Sie den dazu notwendigen Konstruktor und überschreiben Sie die Methode *getArea()*.

c) 1 Punkte

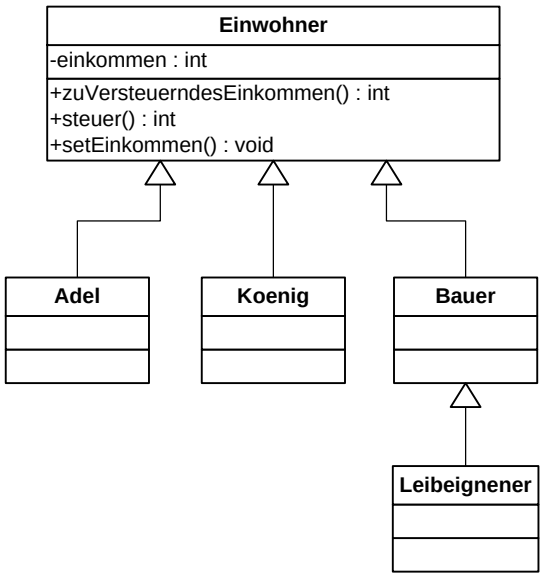
Gegeben sei die folgende *main()*-Methode:

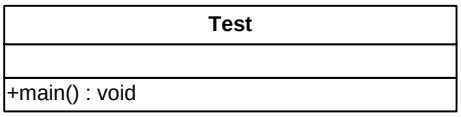
```
Rechteck q1 = new Rechteck(2, 4);
System.out.println(q1.getArea());

Quadrat q2 = q1;
System.out.println(q2.getArea());
```

Wie heißt der objektorientierte „Mechanismus“, der bei *q2.getArea()* zum Tragen kommt?

Aufgabe 2 (10 Punkte)

In einem mittelalterlichen Königreich soll das Finanz- und Steuerwesen auf EDV umgestellt werden. Die verschiedenen Bevölkerungsgruppen werden durch die folgende Klassenhierarchie modelliert:





Das Attribut **`einkommen`** gibt das tatsächliche Jahreseinkommen des Einwohners in Talern an.

Die Methoden **`zuVersteuerndesEinkommen()`** (gibt das zu versteuernde Einkommen zurück) und **`steuer()`** (gibt den Steuerbetrag für ein Jahr zurück) sollen die für jeden Einwohner des Königreiches korrekte Werte gemäß der folgenden königlichen Vorschriften liefern:

- Sofern dieses Gesetz nichts Gegenteiliges aussagt, hat jeder Einwohner sein gesamtes Jahreseinkommen zu versteuern.
- Jeder Einwohner hat 10% seines zu versteuernden Einkommens als Steuer zu entrichten. Der Steuerbetrag wird auf ganze Taler abgerundet, jedoch beträgt die Steuer immer mindestens 1 Taler.
- Der König zahlt auch für sein steuerpflichtiges Einkommen keine Steuern.
- Für Angehörige des Adels beträgt die Steuer mindestens 20 Taler.
- Bei Leibeigenen sind 12 Taler des Jahreseinkommens steuerfrei.

Implementieren Sie die Klassenhierarchie! Überlegen Sie sich zunächst, wie die Methoden in der Klasse **`Einwohner`** implementiert werden müssen. Welche Methoden müssen in den Unterklassen überschrieben werden?
Zusätzlich soll eine Testklasse **`Test`** geschrieben werden, die von jeder nicht-abstrakten Klasse ein Objekt erzeugt, mit einem fiktiven Jahreseinkommen von 100 Talern versieht und das zu versteuernde Einkommen und den Steuerbetrag wieder ausgibt.

Hinweise: Alle Einwohner sind entweder König(in), von Adel, Bauern oder Leibeigene. Gemeinsame Attribute und/oder Methoden sollten in (gemeinsamen) Basisklassen implementiert, Spezialisierungen in den abgeleiteten Klassen durchgeführt werden.

Aufgabe 3 (12 Punkte)

Beim **binärer Heap** handelt sich um eine Datenstruktur, die sich als Vorrangwarteschlange einsetzen lässt, das heißt es können in beliebiger Reihenfolge effizient Elemente mit festgelegter Priorität in den Heap hineingelegt und stets das Element mit höchster Priorität entnommen werden. Die Priorität eines Elements wird durch dessen Schlüssel bestimmt. Über der Menge der Schlüssel muss daher eine totale Ordnung bestehen (siehe „<“-Relation über den ganzen Zahlen).

Ein binärer Heap besteht aus einem Binärbaum, bei dem alle Schichten bis auf die letzte vollständig aufgefüllt sein müssen (→balancierter Baum). Ferner muss die Heap-Bedingung erfüllt sein, d.h. der Schlüssel jedes Kindes eines Knotens ist größer-gleich als der des Knotens selbst. Die Heap-Bedingung garantiert, dass sich an der Wurzel immer ein Element mit minimalem Gewicht befindet.

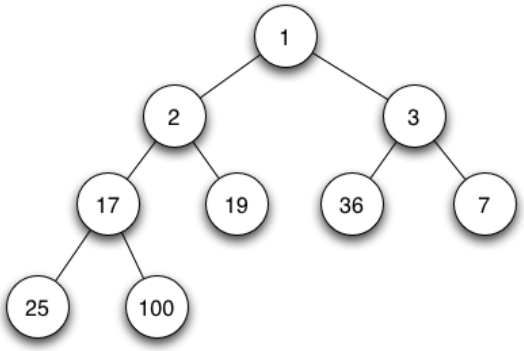


Abbildung 1. Binärer Heap – ein Beispiel

Von Ihnen ist die folgende Schnittstelle (*interface*) zu implementieren:

```
interface HeapTreeI {
      public String search(int key);
      public int countLeaves();
}
```

Die Funktion *search()* sucht den Knoten mit dem angegebenen Schlüssel und gibt dessen Wert zurück. Sollte der Schlüssel nicht enthalten sein, so ist *null* zurückzugeben. Ferner liefert die Funktion *countLeaves()* die Anzahl der Blätter im (Heap-) Baum.

Der Hilfs-Typ *Node* ist wie folgt definiert:

```
class Node {
      int key;
      String value;
      Node left;
      Node right;
}
```