Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

SS 2006
Übungsblatt 7

Informatik II

Zu jeder Aufgabe sind Dateien abzugeben, deren Namen rechts in der Aufgabenüberschrift stehen. Stellen Sie die Dateien in ein extra Verzeichnis (mit beliebigem Namen) und packen Sie dieses zu einem ZIP-Archiv. Geben Sie dieses, wie üblich, per UniWorx ab.

Aufgabe 7-1 — Vererbung (8 Punkte)

Ein *Term* über ganzen Zahlen $\mathbb{Z}$ ist wie folgt definiert: für $z \in \mathbb{Z}$ ist z ein Term, und ein Variablenname v ist ein Term. Sind A und B Terme, dann sind auch $A + B$, $A - B$ und $A * B$ Terme.

a) Geben Sie eine Klassenhierarchie, mit der Sie Terme darstellen können, als UML-Diagramm an. Beachten Sie, daß eine mehrstufige Hierarchie unter Umständen hilfreich sein kann.

b) Implementieren Sie Ihre Klassenhierarchie in Java, und geben Sie eine `toString() : String`-Methode an, die einen Term in lesbarer Form ausgibt.

c) Implementieren Sie eine Methode `evaluate(v : VariableEvaluation) : int`, die einen Term mit einer Variablenbelegung (also einer Funktion Variablenname $\rightarrow \mathbb{Z}$) auswertet. Eine Beispielimplementierung von `VariableEvaluation` finden Sie auf der Vorlesungs-Homepage.

Aufgabe 7-2 — Strategy-Pattern (8 Punkte)

Eine häufig benutzte Anwendung des Strategy-Patterns in der Java-API findet sich in der Implementierung des Sortieralgorithmus für Arrays. Mit der Prozedur `Arrays.sort` der Klasse `java.util.Arrays` wird ein Array sortiert. Für Arrays von `int` oder Objekte, die das Interface `Comparable` implementieren (wie z.B. `java.lang.String`), funktioniert dies direkt:

```
import java.util.Arrays;

public class ComparableExample {
    public static void main(String[] args) {
        Arrays.sort(args);
        System.out.println(Arrays.toString(args));
    }
}
```

Für beliebige andere Objekte wird das Strategy-Pattern eingesetzt. Dabei wird das Interface `java.util.Comparator` verwendet, um eine *totale Ordnung* zu repräsentieren:

Definition 1 ((totale) Ordnung) *Sei M eine Menge. Eine Relation $\preceq \subseteq M \times M$ heisst* Ordnung*, wenn gilt:*

- $\forall x \in M . x \preceq x$ *(Reflexivität)*
- $\forall x, y \in M . x \preceq y \wedge y \preceq x \implies x = y$ *(Antisymmetrie)*
- $\forall x, y, z \in M . x \preceq y \wedge y \preceq z \implies x \preceq z$ *(Transitivität)*

Eine Relation $\prec$ heisst total*, wenn $\forall x, y \in M . x \preceq y \vee y \preceq x$ gilt.*

Wir schreiben $a \prec b$ für $a \preceq b \wedge a \neq b$.

`java.util.Comparator` deklariert eine Methode `int compare(Object o1, Object o2)`, für deren Rückgabewert r gelten muss:

- $r < 0$ genau dann wenn `o1` $\prec$ `o2`
- $r = 0$ genau dann wenn `o1` $=$ `o2`
- $r > 0$ genau dann wenn `o2` $\prec$ `o1`

Ein Array wird dann sortiert, indem man `Arrays.sort(array,comparator)` mit einem Array `array` und einer `Comparator`-Instanz `comparator` aufruft.

a) Die Klasse `java.io.File` repräsentiert ein File im Dateisystem, das entweder ein konkretes File wie z.B. eine Textdatei oder ein Verzeichnis sein kann. Ist ein File-Objekt `f` ein Verzeichnis, bekommen Sie mit `f.listFiles()` die Liste der enthaltenen File-Objekte (als ein Array von `File`-Instanzen). Schreiben Sie drei `Comparator`-Implementierungen, die folgende totale Ordnungen repräsentieren:

   1. eine lexikographische Ordnung bzgl. des Filenamens, also $a \prec b$ gdw. der Name des Files a lexikographisch kleiner ist als der Name des Files b. Beachten Sie hierbei, dass `String` das `Comparable`-Interface implementiert.
   2. eine Ordnung, die Dateigrößen vergleicht.
   3. eine Ordnung, die das Datum der letzten Änderung vergleicht.

   Benutzen Sie die Java-API, um die entsprechenden Methoden von `java.io.File` zu finden:

   `http://java.sun.com/j2se/1.5.0/docs/api/java/io/File.html`

b) Schreiben Sie eine statische Methode, um die Files des aktuellen Verzeichnisses „.“ in den drei Varianten sortiert auszugeben.

Aufgabe 7-3 **Warteschlange** (7 Punkte)

Gegeben sei die folgende Schnittstelle für Warteschlangen:

`IntQueue.java`

```
package queue;

/**
 * Schnittstelle einer Warteschlange, die ganzzahlige Werte aufnehmen kann.
 */
public interface IntQueue {
    /**
     * Wie viele Elemente befinden sich gerade in der Queue?
     */
    public int size();

    /**
     * Stelle einen Wert in die Queue.
     */
    public void enq(int value);

    /**
     * Hole einen Wert aus der Queue
     */
    public int deq();
}
```

a) Schreiben Sie eine Implementierung `SimpleIntQueue` dieser Schnittstelle. Hinweise:

   - Eine Instanz dieser Klasse soll eine Kapazität haben, die während der gesamten Lebensdauer unverändert bleibt. Diese Kapazität soll dem Konstruktor als Argument übergeben werden.
   - Überprüfen Sie, wo nötig, Verbedingungen per `assert`.

- Die Warteschlangen-Elemente sollen in einem `int`-Array abgelegt werden.

b) Schreiben Sie eine Unterklasse `ResizableIntQueue` von `SimpleIntQueue`, die die Methode `enq()` so überschreibt, dass die Kapazität automatisch verdoppelt wird, wenn die Warteschlange voll ist.

c) Testen Sie Ihre Implementierungen mit der Klasse `QueueTester` (herunterzuladen von der Informatik-Homepage).

Abgabe: Per UniWorx, bis spätestens Montag, den 26.6.2006 um 9:00 Uhr.