Informatik II (D-ITET)
FS 2009
Prof. Dr. F. Mattern
Philipp Bolliger (bolligph@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Verteilte Systeme
http://www.vs.inf.ethz.ch

Übungsserie Nr. 5

Ausgabe: 25. März 2009
Abgabe: 01./02. April 2009

*Laden Sie sich für die folgenden Aufgaben die Code-Fragmente **IntList.java** und **TestIntList.java** von der Vorlesungswebseite herunter. Legen Sie die beiden Dateien in ein Verzeichnis. Implementieren Sie alle Aufgaben in der Klasse `IntList` und testen Sie die Funktionalität der Methoden mittels der einzigen Methode `main` in der Klasse `TestIntList`.*

1. Aufgabe: (6 Punkte) Implementierung einer verketteten Liste

Vervollständigen Sie die Klasse `IntList`, die `int`−Werte als einfach verkettete, unsortierte Liste verwaltet. Beachten Sie dazu auch Folie 173 ff. des Vorlesungsskripts. Die Elemente der Liste sind Instanzen der Klasse `IntListElem`, welche neben einem `int`−Wert (dem eigentlichen Inhalt) eine Referenz auf das nächste Listenelement speichern. Der Anfang der Liste (die Referenz auf das erste Listenelement, auch *Header* genannt) ist in der Klasse `IntList` in der *privaten* Instanzenvariablen `first` gespeichert. Die Methoden `empty()` und `addFirst(int i)` sind bereits implementiert. Vervollständigen Sie die Methoden:

(1a) *(1 Punkt)*

```
public void addLast(int i)
```

zum Einfügen eines neuen Elementes i am Ende der Liste,

(1b) *(2 Punkte)*

```
public int removeFirst()
public int removeLast()
```

zum Entfernen des ersten bzw. des letzten Elementes und Rückgabe dessen Wertes und

(1c) *(1 Punkt)*

```
public void print()
```

zur Ausgabe der Werte der Listenelemente (durch Leerzeichen getrennt), beginnend mit dem ersten Element.

(1d) *(2 Punkte)* Testen Sie die Methoden der obigen Teilaufgaben durch die jeweils nötigen Anweisungen in der `main`-Methode der Klasse `TestIntList`. Laden Sie sich dazu das Code-Fragment `TestIntList.java` von der Vorlesungswebseite herunter und vervollständigen Sie es.

Beachten Sie, dass bisher alle Methoden der Klasse `IntList` mit den Variablen einer Listen-Instanz arbeiten und deshalb nicht als `static` deklariert werden sollen und können.

2. Aufgabe: (8 Punkte) Liste mit Zufallszahlen

(2a) *(2 Punkte)* Fügen Sie zur Klasse `IntList` die Methode

```
public void addRandomElements(int n)
```

hinzu, die, am Ende der bestehenden Liste, n neue Elemente anfügt. Die Werte der neuen Einträge sollen zufällig aus dem Bereich 0 bis 400 stammen. (Pseudo-)Zufallszahlen können (und müssen) mit den Methoden der Klasse `Random` aus dem Java-Paket `java.util` erzeugt werden (vgl. auch Aufgabe 2 der Übungsserie 2).

(2b) *(2 Punkte)* Fügen Sie nun zur Klasse `IntList` die Java-Methode

```
public static IntList createRandomList(int n)
```

hinzu, die eine *neue* Listen-Instanz erstellt und diese mit n Zufallszahlen auffüllt (Sie können in der Methode natürlich die obige Methode benutzen, Sie müssen nur überlegen *wie*).

Beachten Sie, dass diese Methode jetzt instanzunabhängige Funktionalität bereitstellt und deshalb als `static` deklariert ist. Sie können nun aus Ihrer Testmethode `main` in der Klasse `TestIntList` z.B. mit

```
IntList myRandomList = IntList.createRandomList(n)
```

aufrufen, da `static`-Methoden instanzunabhängig sind.

(2c) *(2 Punkte)* Zählen Sie nun die Anzahl der Listeninstanzen, die im Programm jemals erstellt worden sind, mittels einer privaten `static`-Variable in der Klasse `IntList`. Diese existiert nur einmal für alle Instanzen. Diese Variable soll bei jedem `new IntList()` erhöht werden. Überlegen Sie, wo Sie dieses am besten (innerhalb von `IntList`) implementieren können.

(2d) *(2 Punkte)* Testen Sie die Methoden der obigen Teilaufgaben durch die jeweils nötigen Anweisungen in der `main`-Methode der Klasse `TestIntList`. Vervollständigen Sie dazu das Fragment `TestIntList.java` weiter.

3. Aufgabe: (10 Punkte) Insertion-Sort

(3a) *(4 Punkte)* Betrachten Sie folgenden *Insertion-Sort*-Algorithmus zum Sortieren einer Liste. Wir konstruieren aus der alten unsortierten Liste eine neue, sortierte Liste:

```
Erstelle neue leere Liste
Solange alte Liste noch nicht leer ist {
    Entferne das erste (das letzte) Element aus der Liste
    Fuege es in die neue Liste an der richtigen Stelle ein
}
```

Vervollständigen Sie die Methode `sort()`, welche eine in Aufgabe *2a* oder *2b* konstruierte Liste mittels *Insertion-Sort* aufsteigend sortiert und die sortierte Liste zurückgibt. Die ursprüngliche Liste ist danach leer.

(3b) *(1 Punkt)* Wie könnten Sie `sort()` jetzt so verändern, dass die alte Listen-Instanz selbst auf die sortierte Liste zeigt und kein Rückgabewert erforderlich ist? Die Lösung können Sie in einer zusätzlichen Methode `void sortSelf()` implementieren, Sie können darin natürlich Ihre bereits existierende `sort`-Methode benützen.

(3c) *(2 Punkte)* Testen Sie das Sortieren in der Testmethode `main` der Klasse `TestIntList`. Vervollständigen Sie dazu das Fragment `TestIntList.java` weiter.

(3d) *(1 Punkt)* Ist die Methode `sortSelf()` besser als die Methode `sort()`? Wieso? Was heisst *besser*?

(3e) *(2 Punkte)* Was müssten Sie an den Klassen `IntList` und `IntListElem` ändern, wenn Sie anstelle von `int`-Werten `Strings` verwalten und sortieren wollen?

Summe: 24 Punkte