KV Software Engineering
Übungsaufgaben SS 2005

Martin Glinz, Silvio Meier,
Nancy Merlo-Schett, Katja Gräfenhain

Übung 3

Aufgabe 1 (8 Punkte) – Vertragsorientierte Schnittstelle und Vererbung

Gegeben ist die Schnittstelle der Java-Klasse *List* zur Verwaltung einer Liste von Integer-Werten und die Schnittstellen der Subklassen *UniqueList*, *AscendingSortedList* und *DescendingSortedList*. Die Klasse *UniqueList* stellt eine Liste zur Verfügung, welche keinen Wert innerhalb der Liste zweimal zulässt. Dies wird über den Vertrag der Klasse erzwungen. Die Klasse *AscendingSortedList* stellt eine Liste zur Verfügung, welche eine Liste von aufsteigend sortierten Integer-Werten verwaltet. Die Sortierung erfolgt, indem beim Einfügen eines Elements mit der Methode *addElement(int element)*, das Element an der richtigen Stelle einsortiert wird. Die Klasse *DescendingSortedList* stammt von der Klasse *AscendingSortedList* ab und stellt eine Liste von Integer-Werten zur Verfügung, welche absteigend sortiert ist. Die Klasse funktioniert ansonsten analog wie *AscendingSortedList*.

Hinweise: Im Nachfolgenden bedeutet { ... }, dass die betreffende Methode eine Implementierung hat, diese aber hier nicht gezeigt wird. Für die bessere Verständlichkeit wird bei der Formulierung von Gleichheit in Ausdrücken von Verträgen das doppelte Gleichheitszeichen (==) verwendet. INV bedeutet Invariante,

ansonsten gilt die Notation aus dem Vorlesungs-Skript für die Formulierung von Verträgen.

```
// INV: -
class List {
       // Hier werden die Elemente gespeichert
       int[] elements = null;

        // PRE:  -
        // POST: elements <> null and size() == 0
       public List() { ... }

        // PRE:  -
        // POST: size()@PRE + 1 == size() and
        //     elementExists(element)
       public abstract void addElement(int element);

        // PRE:  0 <= index and index < size()
        // POST: result == elements[index]
       public int getElementAt(int index)  { ... }

        // PRE:  -
        // POST: result >= 0 and
        //      result == elements.length
       public int size() { ... }

        // PRE:   -
        // POST:  if there exists k >= 0 and k < size() - 1:
        //     elements[k] == element then result == true
        //     else result == false endif
       public boolean elementExists(int element) { ... }
}

// INV:  -
class UniqueList extends List {

        // PRE  not elementExists(element)
        // POST size()@PRE + 1 == size() and
        //     elements[size() - 1] == element
       public void addElement(int element) { ... }
}

// INV:  if size() > 1 then for all k >= 0 and k < size() - 1:
//     elements[k] <= elements[k + 1]
//     else true endif
class AscendingSortedList extends List {

        // PRE  -
        // POST size()@PRE + 1 == size() and elementExists(element)
       public void addElement(int element) { ... }
}

// INV: if size() > 1 then for all k >= 0 and k < size() -1:
// elements[k] >= elements[k+1]
// else true endif
class DescendingSortedList extends AscendingSortedList {

       // PRE:  -
       // POST: size()@PRE + 1 == size() and elementExists(element)
       public void addElement(int element) { ... }
}
```

Es gibt drei Arten von Vererbung: Steinbruchvererbung, Spezialisierungsvererbung und Substitutionsvererbung. Beantworten Sie folgende Fragen im Zusammenhang mit den gezeigten Schnittstellen:

a.) Beschreiben Sie die Bedingungen, welche im Allgemeinen für eine Substitutionsvererbung erfüllt sein müssen. (2P)

b.) Beschreiben und begründen Sie **kurz**, um welche Art es sich bei der Vererbung zwischen

- *List* und *UniqueList* (2P)
- *List* und *AscendingSortedList* (2P)

handelt

c.) Beschreiben Sie und begründen Sie **kurz**, um welche Art der Vererbung es sich zwischen A*scendingSortedList* und *DescendingSortedList* handelt (2P).

Aufgabe 2 (12 Punkte) – Zusammenarbeit und Vertragsformulierung

Untersuchen Sie das Codebeispiel des Bankensystems in der beiliegenden Datei "Bank.pdf" bzw. "Bank.zip".

Um
(1) welche Formen der Zusammenarbeit (Leistungserbringung, Informationsaustausch, Informationsteilhabe),
(2) welche Mittel,
(3) (falls vorhanden) welches Prinzip (Informationsaustausch: Bringprinzip, Abonnementsprinzip, Holprinzip),
(4) (falls vorhanden) welche Wirkung auf den Systemzustand (unverändert / verändert)
handelt es sich, wenn die folgenden Methodenaufrufe ausgeführt werden? Erläutern Sie kurz (1 Satz) die Ausgangslage und was genau passiert.

a) In der Klasse Account, in der Methode *accumulateInterests()* der Methodenaufruf notifyObservers(new Float(oldBalance) (2 Punkte)

b) In der Klasse Account**,** in der Methode *accumulateInterests*(), der Methodenaufruf calculateInterests() (1 Punkt)

c) In der Klasse Bank**,** in der Methode *addBankAccount(Account a)* das Statement currentAccountBalance += a.balance (1 Punkt)

d) Finden Sie eine Zusammenarbeit in Form der Informationsteilhabe? Begründen Sie Ihre Antwort **kurz** (1-2 Sätze). Falls Ihrer Meinung nach eine Informationsteilhabe vorhanden ist, wo und welche Klassen sind daran beteiligt? (1 Punkt)

e) Erstellen Sie die fehlenden Verträge für die folgenden Methoden. Halten Sie sich an die Notation aus dem Vorlesungsskript. Verwenden Sie das doppelte Gleichheitszeichen (==) für die Formulierung von Gleichheit. (7 Punkte)

- `Bank.addBankAccount(Account a)` (1 Punkt)
- `Bank.removeAccount(Account a)` (1 Punkt)
- `Bank.removeAccount(int accountNo)` (1 Punkt)
- `Bank.transferAmount(int srcAccount, int destAccount, float amount)` (2 Punkte)
- `Bank.depositMoney(int accountNo, float amount)` (1 Punkt)
- `Bank.withdrawMoney(int accountNo, float amount)` (1 Punkt)