ALP II SS 2012
Prof. Dr. Margarita Esponda

11. Übungsblatt

Ziel: Auseinandersetzung mit Dynamischen Datenstrukturen und deren verschiedenen Implementierungen.

1. Aufgabe (6 Punkte)

a) Was verstehen Sie unter Vererbungspolymorphie? Erläutern Sie Ihre Antwort mit einem einfachen Beispiel.

b) Was sind generische Klassen? Wozu sind sie gut?

c) Wann genau sollen ***Exceptions*** (Ausnahmefehler) programmiert werden?

d) Was ist der Unterschied zwischen ***Exceptions***, ***RuntimeExceptions*** und ***Errors*** in Java?

e) Was sind Iteratoren? Wozu sind sie gut?

f) Wann ist es sinnvoll, innere Klassen zu programmieren? Geben Sie ein Beispiel.

2. Aufgabe (20 Punkte)

Programmieren Sie eine **SortedLinkedList**-Klasse, die (analog zum Vorlesungsbeispiel mit Binärsuchbäumen) Schlüssel und das damit verbundene Daten-Objekt speichert.

Bei der Implementierung soll eine doppelt verkettete Liste verwendet werden und die Daten sollen nach den Schlüsseln in sortierter Reihenfolge gespeichert werden.

Folgende Schnittstelle soll dabei implementiert werden:

```
public interface List<K extends Comparable <K>, D>{

    /* It checks whether a key exists in the list or not */
    public boolean contains(K key);

    /* get the data associated with a key, or null if the key does not exist.*/
    public D getData(K key);

    /* a key and the associated data object are sorted into the list */
    public void store(K key, D data);

    /* removes the ListNode-Object which contains the key */
```

```
        public D remove(K key);

        /* checks if the list is empty */
        public boolean empty();

        /* returns the size of the List */
        public int size();

        /* removes all items from the List-Object */
        public void clear();

        /* generates a string from the List-Object (this) */
        public String toString();
}
```

Wichtige Hinweise:

a) Verwenden Sie **Dummy-ListNode**-Objekte, um den Algorithmus der **store**-/**remove**-Operationen möglichst einfach zu gestalten (siehe Vorlesungsfolien).

b) Die **ListNode**-Klasse soll als innere Klasse innerhalb der **SortedLinkedList**-Klasse definiert werden.

3. Aufgabe (10 Punkte)

Erweitern Sie Ihre Implementierung um folgende zwei Methoden:

```
        /* returns an Iterator-Object for the SortedList-Object */
        public Iterator<K> iterator();
        /* merges the invoking SortedLinkedList-Object (this) with the list-Object
        * of the argument*/
        public void merge(SortedLinkedList <K, D> list);
```

Um die **Iterator**-Objekte zu erzeugen soll eine **ListIterator**-Klasse als innere Klasse implementiert werden, die folgende Schnittstelle implementiert:

```
public interface Iterator <K> {
    /* looks if a next ListNode-object after a current ListNode exists */
    public boolean hasNext();

    /* returns the next key in the list and advances the current position */
    public K next();
}
```

4. Aufgabe (8 Punkte) Freiwillige Aufgabe

a) Vervollständigen Sie die **BinarySearchTree**-Klasse aus der Vorlesung mit den in der Vorlesung besprochenen Operationen (Methoden):

```
public boolean empty();
public TreeNode min(TreeNode node);
public TreeNode max(TreeNode node);
public TreeNode succ(TreeNode node);
public TreeNode pred(TreeNode node);
public boolean delete(T key);
public int deep();
```

b) Ist Ihre **delete**-Operation kommutativ (in dem Sinne, dass die hinterlassenen Bäume gleich aussehen, unabhängig von der Reihenfolge der durchgeführten delete-Operationen)? Erläutern Sie Ihre Antwort.

Alle Methoden der Klassen-Implementierungen sollen ausführlich getestet werden.