Übungsblatt 9-1 – Lösungen

Die Aufgaben auf diesem Übungsblatt befassen sich mit der Nutzung und der Deklaration von Iteratoren. Die Aufgaben- geben dabei Coderahmen vor, die passend ausgefüllt werden sollen. Diese Art der Aufgabenstellung entspricht den Aufgaben, die in der Klausur gestellt werden. Die vorgegebenen Codrahmen sollen zugleich eine Lösungsidee vorgeben, einen bestimmten Lösungsweg erzwingen und den Schreibaufwand reduzieren.

Entwurfsmuster Iterator

Die Klassen `Data` und `DataIterator` besitzen Konstruktoren und weitere Methoden, die hier aber nicht verwendet werden sollen.

```
public class Data {
    // ... (Konstruktor und weitere Methoden sind nicht von Interesse)

    public Iterator iterator() {
        return new DataIterator();
    }

    private class DataIterator extends Iterator {
        // ... (Konstruktor und weitere Methoden sind nicht von Interesse)
    }
}

public abstract class Iterator {
    public abstract boolean hasNext();
    public abstract Object next();
}
```

Die Methode `int countFirst( Data structure )` soll die Häufigkeit zurückgeben, mit der das erste durch den Iterator gelieferte Objekt in `structure` vorkommt. Gibt es kein Objekt, das durch den Iterator geliefert wird, so soll der Wert `0` zurückgegeben werden. Die Gleichheit soll mit der Methode `equals` bestimmt werden.

```
public class Use
{
    public static int countFirst( Data structure )
    {
        int count = 0;
        Iterator it = structure.iterator();
        if ( it.hasNext() )
        {
            Object ref = it.next();
            count = 1;
            while ( it.hasNext() )
            {
                if ( it.next().equals(ref))
                 {
                     count++;
                 }
            }
        }
        return count;
    }
```

Entwurfsmuster Iterator

Die Klasse `NLists` verwaltetet `n` Listen in einer Liste. Die Klasse `NLists` soll einen Iterator bereitstellen, der die Inhalte aller `n` Listen nacheinander bereitstellt. Ergänze den vorgegebenen Code passend. In der bekannten Klasse `DoublyLinkedList` stehen u.a. die Methoden `size`, `get` und `iterator` zur Verfügung.

```
public class NLists
{
    DoublyLinkedList lists;

    public NLists( int n ) {
        lists = new DoublyLinkedList();
        for ( int i = 0; i < n; i++ ) {
            lists.add( new DoublyLinkedList() );
        }
    }

    public void add( int listNo, Object content ) {
        if ( listNo < lists.size() ) {
            lists.get( listNo ).add( content );
        }
    }

    public Iterator iterator() {
        return new NListsIterator();
    }

    private class NListsIterator extends Iterator
    {
        Iterator currentIterator;     // kann den Iterator einer Liste referenzieren
        int currentListNo;            // kann eine der Listen identifizieren

        public NListsIterator() {
            currentListNo = 0;
            currentIterator = null;
            if ( lists.size() > 0 ) {
                currentIterator = lists.get( 0 ).iterator();
            }
        }

        public boolean hasNext() {
            while ( currentListNo < lists.size() ) {
                if ( currentIterator.hasNext() ) {
                    return true;
                } else {
                    currentListNo++;
                    if ( currentListNo < lists.size() ) {
                        currentIterator = lists.get( currentListNo ).iterator();
                    }
                }
            }
            return false;
        }

        public Object next() {
            if ( currentIterator.hasNext() ) {
                return currentIterator.next();
            } else {
                throw new IllegalStateException();
            }
        }
    }
}
```