Homework 3

Problem 1 [1.5 point]

Write a class **SimpleArrayList**, objects of which represent vectors, i.e., "extensible arrays" of ints (similar to ArrayList in Java or vector in C++).

The class declares the following fields

• **size** — current number of elements in the vector;

• **cap** — current capacity (size) of the array which holds the elements; size of them are "true" elements which have been put there by the user and the remaining provide room for elements which will be put later (in this way we avoid allocating a new array every time we want to add an element);

• **arr** — reference to a ordinary array of size cap where the elements of the vector are held;

• **INITIAL_CAPACITY** — static constant determining the initial capacity of the vector (initial size is, of course, zero).

Define also member functions

• Default constructor, creating a vector of size (size) zero and the capacity (cap) equal to **INITIAL_CAPACITY**.

• Constructor taking one value of type int and creating a vector with this single element (size will be 1).

• Constructor taking an array of ints; its elements will become elements of the vector, size will be the number of these elements and the capacity must be appropriately chosen (see below).

• Constructor taking another vector of type **SimpleArrayList**: its elements will be elements of the vector being created.

• Method **size** returning the size (size) of the vector.

• Method **clear** „clearing" the vector; after this operation the state of the vector becomes identical to the state of an object created by the default constructor.

• Method **trim**: after calling this method the array arr has capacity equal to the current size.

• Method **insert** taking an index (say, ind) and an array of ints (say, other). The method:

throws an **IndexOutOfBoundsException** if the value of ind is larger than size or negative;

inserts elements from the array other (which is of size, say, sz) into the vector starting at position indicated by the index ind. If the current capacity is sufficient, existing elements of the vector from position **ind** are shifted to the right to make room for new elements from the array other. If the capacity is too small to accommodate old and new elements, a new array **arr** is allocated, with capacity twice as big as necessary, i.e., 2*(size+sz). Old and new elements are copied into this array at correct positions.

The method returns the reference to the object that it was invoked on (this).

• Methods

- public SimpleArrayList **insert**(int ind, int e)
- public SimpleArrayList **append**(int e)
- public SimpleArrayList **append**(int[] a)
- public SimpleArrayList **append**(SimpleArrayList a)

which:

– the first inserts a single element **e** at position **ind**;

– the second appends a single element **e** at the end of the vector;

– the third appends to the vector elements from the array passed as an argument;

– the fourth appends to the vector elements from another vector passed as an argument.

All these methods should be very **short**; they should just use the first version of **insert**, described above.

• **Methods**

public int **get**(int ind)

public SimpleArrayList **set**(int ind, int val)

The first of them returns ind-th element of the vector, while the second modifies the element under index ind assigning the value val to it. Both throw **IndexOutOfBoundsException** for invalid values of the index ind.

• **Method**

public String **toString**()

which returns a string representation of the vector (it will be called automatically when the reference to an object of the class is passed to method println).

Note: function System.arraycopy should be used for copying arrays or parts of arrays.

For example, the program

```
public class SimpleArrayList {
    private final static int INITIAL_CAPACITY = 5;
    private int cap   = INITIAL_CAPACITY;
    private int size  = 0;
    private int[] arr = new int[cap];

    // ... constructors and methods

    public static void main(String[] args) {
        SimpleArrayList a =
            new SimpleArrayList()
```

```
                .append(new int[]{1,3}).insert(1,2)
                .append(6).insert(3,new int[]{4,5});
        SimpleArrayList b = new SimpleArrayList(a);
        for (int i = 0; i < a.size(); ++i)
            a.set(i,a.get(i)+6);
        b.append(a).append(13).trim();
        System.out.println("a -> " + a);
        System.out.println("b -> " + b);
    }
}
```

should print

```
a -> Cap=12, size=6: [ 7 8 9 10 11 12 ]
b -> Cap=13, size=13: [ 1 2 3 4 5 6 7 8 9 10 11 12 13 ]
```

*Do **not** use any classes except those in the package java.lang.*

Problem 2 [1 points]

Write a program which checks correctness of brackets (round, curly and square) in a text file (or just a multi-line string). This can be done by reading the text character by character and:

• if an **opening bracke**t has been encountered, put information on its kind (round, curly or square) onto a stack;

• if a **closing bracket** has been encountered (not necessarily in the same line as the corresponding opening bracket), pop information on the kind of the last opening bracket off the stack and check if it matches the kind of the closing bracket encountered.

Implement the stack by creating class **Node**, as in implementation of singly-linked lists. Objects of type **Node** represent data pushed onto the stack (kinds of brackets, as a value of type char, String, int . . .) and contain a reference next to the next node, as it is usual in singly-linked lists. Note that stack is a **singly-linked** list where adding and removing elements is always performed at the beginning, so traversing the list is never needed.

The program reads the input and prints "OK" if all **brackets** are correct. If an error is detected, the program describes it, prints line number and the offending line itself indicating the location of the error by ^ character under the incorrect bracket. After reading the whole input, the program checks if the stack is **empty**; if not, it prints information about that and a list of all brackets that have not been closed.

For example, for a file

```
Warsaw(){
London[xxxx} (
Madrid Paris)}
Berlin
```

the program should print something like

```
London[xxxx} (
          ^
ERROR in line 2. '}' found, but ']' expected.
```

Problem 3[0.5 points]

Write a function which takes a position of a chess knight on the chessboard. Position is specified as a string of length 2: the first character denotes the column, or file (from 'a' through 'h') and the second the row, or rank (from '1' through '8'). The function returns a string containing, space separated, positions that are available for the knight in one move.

For example, the following program

```
public class ChessKnight {
    public static String knightMoves(String pos) {
        // ...
    }

    public static void main (String[] args) {
        for (String s : new String[]{"A1","d5","g6","C8"})
            System.out.println(s + " -> " + knightMoves(s));
    }
}
```

should print

```
A1 -> b3 c2
d5 -> b4 b6 c3 c7 e3 e7 f4 f6
g6 -> e5 e7 f4 f8 h4 h8
C8 -> a7 b6 d6 e7
```

Problem 4 [1 points]

Create a class **MyString**, objects of which describe strings of characters, and have only one private field **str** of type String. In the class define:
• default (parameterless) constructor, which sets the value of str to empty string (but not **null**);
• constructor taking a String; the value of the argument becomes the value of str;
• method getLength() returning the length of the current value of str;
• method getChar(int n) returning n-th character (zero based) of str. If the argument is illegal, the method throws (unchecked) exception **IllegalArgumentException**;
• method **append**(String s) appending s to str;
• method **append**(int rep, String s) appending rep repetitions of s to str;
• method **prepend**(String s) prepending s to str;
• method **insert**(int pos, String s) inserting s into str at position pos; for example, if current str is "abcdef", then after inserting "123" at position 2, one should get "ab123cdef". If the argument is illegal, the method throws (unchecked) exception **IllegalArgumentException**;
• method reset(String s) which substitutes s for str;
• redefinition of the **toString** method from class Object.

Do not use any classes from packages other than java.lang. Remeber that objects of class String are immutable!