ITI 1221. Introduction to Computer Science II Summer 2015

Assignment 2

Deadline: Monday July 6, 23:59 (strongly suggested to hand-in by July 3rd)

Objectives

- Further understanding of interfaces
- Introduction to the applications of stacks
- Introduction to the implementation of stacks

Introduction

This assignment is about three implementations of the interface Stack as well as an application of a Stack.

1 Modifying the interface Stack: adding a method clear() [5 marks]

Modify the interface Stack below adding an abstract method public void clear().

```
public interface Stack {
    public abstract boolean isEmpty();
    public abstract Object peek();
    public abstract Object pop();
    public abstract void push( Object element );
}
```

Click here: Stack.java.

2 Implementing the method clear() in the class ArrayStack [5 marks]

The class ArrayStack uses a fixed size array and implements the interface Stack. Now that the interface Stack has been modified to have a method clear(), the current implementation of the class ArrayStack is broken (try compiling it without making any change).

Since the class ArrayStack implements the interface Stack, it has to provide an implementation for all the methods that are declared by the interface. Consequently, write an implementation for the method void clear(). It removes all of the elements from this ArrayStack. The stack will be empty after this call returns.

Click here: ArrayStack.java.

3 Using a stack: Calculator [30 marks]

In this question, there is a simple stack-based language to evaluate arithmetic expressions. The language for this question is actually a sub-set of a language called PostScript, which is a file format often used with printers. The main data-structure used by a PostScript interpreter is a stack. For the interpreter presented in the next pages, you must implement the operations add, sub, exch, dup, and pstack. Here are the 5 instructions of the language.

add:
: pops off the top two elements from the operands stack, adds them together and pushes back the result onto the stack;

sub:
: pops off the top two elements from the operands stack, subtracts them together and pushes back the result onto the stack. E.g.: (3 - 1) would be represented as “3 1 sub”;

exch:
: exchanges the order of the two elements on the top of the stack.

dup:
: duplicates the top element of the stack;

pstack:
: prints the content of the stack. It is important that the content of the stack remains the same after a call to the instruction pstack. The operation must not destroy or modify the content of the stack. Use the format of the example below.

The execution of the following PostScript program,

```
> java Run "3 pstack dup pstack add pstack"
```

produces the following output,

```
-top-
INTEGER: 3
-bottom-
-top-
INTEGER: 3
INTEGER: 3
-bottom-
-top-
INTEGER: 6
-bottom-
```

The class Calculator is an interpreter for the language. The implementation requires three additional classes: an implementation of a Stack, Token and Reader.

```
public class Calculator {

   private Stack operands;

   public Calculator() {
      operands = new ArrayStack( 100 );
   }
   public void execute( String program ) {
      Reader r = new Reader( program );
      while ( r.hasMoreTokens() ) {
         Token t = r.nextToken();
```

```
            if ( ! t.isSymbol() ) {
                operands.push( t );
            } else if ( t.sValue().equals("add") ) {
                // the implementation of the operation "add"
            } else if ( t.sValue().equals("sub") ) {
                // the implementation of the operation "sub"
            } else if ( t.sValue().equals("exch") ) {
                // the implementation of the operation "exch"
            } else if ( t.sValue().equals("dup") ) {
                // the implementation of the operation "dup"
            } else if ( t.sValue().equals("pstack") ) {
                // the implementation of the operation "pstack"
            }
        }
    }
}
```

The input for the method execute is a string that contains a valid PostScript program, e.g. "1 pstack dup pstack". The input is parsed into tokens by the Reader. For the current example, the first call to the method nextToken() returns a token that represents the 1, the second call returns a token that represents the symbol pstack, and so on. The tokens that are not symbols are pushed onto the stack whilst the symbols are immediately evaluated. Here is a brief description of the classes Token and Reader.

Token:

A token is an immutable object that represents either a boolean value, an integer or a symbol.

```
class Token {
    private static final int BOOLEAN = 0;
    private static final int INTEGER = 1;
    private static final int SYMBOL = 2;

    private boolean bValue;
    private int iValue;
    private String sValue;
    private int type;

    Token( boolean bValue ) {
        this.bValue = bValue;
        type = BOOLEAN;
    }
    Token( int iValue) {
        this.iValue = iValue;
        type = INTEGER;
    }
    Token( String sValue ) {
        this.sValue = sValue;
        type = SYMBOL;
    }
```

```
        public boolean bValue() {
            if ( type != BOOLEAN )
                throw new IllegalStateException( "not a boolean" );
            return bValue;
        }
        public int iValue() {
            if ( type != INTEGER )
                throw new IllegalStateException( "not an integer" );
            return iValue;
        }
        public String sValue() {
            if ( type != SYMBOL )
                throw new IllegalStateException( "not a symbol" );
            return sValue;
        }
        public boolean isBoolean() { return type == BOOLEAN;  }
        public boolean isInteger() { return type == INTEGER;  }
        public boolean isSymbol()  { return type == SYMBOL;   }
        public String toString() {
            switch (type) {
            case BOOLEAN: return "BOOLEAN: " + bValue;
            case INTEGER: return "INTEGER: " + iValue;
            case SYMBOL:  return "SYMBOL: "  + sValue;
            default:      return "INVALID";
            }
        }
    }
```

Reader:

A reader parses the input string into Tokens.

```
    class Reader {
        private StringTokenizer st;

        Reader( String s ) {
            st = new StringTokenizer(s);
        }
        public boolean hasMoreTokens() {
            return st.hasMoreTokens();
        }
        public Token nextToken() {
            String t = st.nextToken();

            if ( "true".equals( t ) )
                return new Token( true );

            if ( "false".equals( t ) )
                return new Token( false );
            try {
               return new Token( Integer.parseInt( t ) );
            } catch ( NumberFormatException e ) {
               return new Token( t );
            }
         }
      }
```

Run:

contains the main method.

```
public class Run {
   public static void main( String[] args ) {
      Calculator c = new Calculator();
      for ( int i=0; i<args.length; i++ ) {
         c.execute( args[ i ] );
      }
   }
}
```

For this question, you must complete the implementation of the 5 instructions (add, sub, exch, dup and pstack) in the class Calculator.

4 Implement the class DynamicArrayStack (by modifying ArrayStack) [25 marks]

The class ArrayStack has a severe limitation, its capacity is fixed. In class, we have seen an implementation technique called "dynamic array" that allows to circumvent this limitation.

Create a new class called DynamicArrayStack. For this, you will use the class ArrayStack as a starting point. Simply copy the file ArrayStack.java to a new file called DynamicArrayStack.java. Make all the necessary changes so that it can be compiled. In particular, change the name of the class to DynamicArrayStack and change the name of the constructor accordingly. Make sure that you have a valid starting point (i.e. this new class can be compiled).

- Modify the implementation of the method void push(Object element) so that the size of the array is increased when the array is full.

  In Java you can't actually increase the size of an array, so what you have to do is allocate a new array which is bigger than the existing one, copy all the elements from the old array into the new one, set the instance variable to point at the new array, and then proceed adding the element.

  Several strategies are possible to increase the size of the array. Here, you will be implementing the following one. The initial size of the stack is determined by the parameter of the constructor of the class DynamicArrayStack(int capacity). Every time the array will be full, the method push will create an array that has 1.5 times its current capacity.

- Modify the implementation of the method Object pop() so that the size of the array is decreased when the number of elements currently stored in the array is less than half of its current size. The size of the new array should be either its current size or the initial capacity of the DynamicArrayList when it was first created (which ever size value is greater);

- The method clear removes all of the elements from this DynamicArrayStack. The stack will be empty after this call returns. Furthermore, the size of array used to store the elements must equals to the capacity of the DynamicArrayStack when it was first created.

5 Creating a LinkedStack [30 marks]

In class, we have seen a new technique for implementing data structures that has the following benefits:

- has a variable size (does not have a fixed capacity);
- always uses an amount of memory that is proportional to the number of elements stored in the stack (does not waste any memory);
- is efficient (elements are never copied from one array to another).

Create an implementation for the interface Stack using linked elements (the lecture notes contain most of the implementation details). Besides implementing the methods of the interface Stack, the LinkedStack must implement the instance method booleean equals(Object o).

- boolean isEmpty(): returns true if this LinkedStack is empty; and false otherwise;
- Object peek(): returns a reference to the top element; does not change the state of this LinkedStack;
- Object pop(): removes and returns the element at the top of this LinkedStack;
- void push(Object element): puts an element onto the top of this LinkedStack;
- boolean equals(Object other): returns true other if designates a Stack (any implementation of the interface Stack) that contains the same number of elements as this stack, and each element of the other stack must be “equals” to the corresponding element in this stack. Returns false otherwise. Both stacks must not be changed by the method equals, i.e. before and after a call to the method equals, this and the other stack must contain the same elements, in the same order.

6 Rules and Regulations [5 marks]

- Please follow the general directives for assignments.
- You must abide by the following submission instructions:

Include all your java files into a director called **u1234567** where 1234567 is your student id.
Zip this directory and submit via blackboard.
The directory must contain the following files (with your implementation):

- README.txt
- Stack.java
- ArrayStack.java
- Calculator.java
- Token.java
- Reader.java
- Run.java
- DynamicArrayStack.java

- LinkedStack.java
- Test1.java
- Test2.java
- Test3.java
- Test4.java

- The assignment is individual (plagiarism or collaborations towards the solution of the assignment will not be tolerated).

Read the information on academic fraud below.

NOTES:

1. Note on implementing the operations "add" and "sub" for the Calculator.
   Every time the stack will contain only objects of type Token; as you can see in part of the startup code given:
   ```
   Token t = r.nextToken();
   operands.push( t );
   ```
   So, every time you pop() you know you will be getting back an object of class Token and that it should represent an integer. You need to look inside class Token for methods useful to retrieve the value of this integer.

   Hint: at all times, the stack should only contain Tokens, this will simplify the implementation (this implies that the objects added onto the stack will be Tokens).

2. For the method equals of the class LinkedStack, it would seem to need special cases for situations where **other** designates an ArrayStack, a LinkedStack or an ArrayStack. This is not the case! Read the question carefully, **other** may designate any valid implementation of a Stack (including UnknownStack, as long as UnknownStack is a valid implementation of the interface Stack). You cannot assume a particular implementation.

Academic fraud

This part of the assignment is meant to raise awareness concerning plagiarism and academic fraud. Please read the following documents.

- http://www.uottawa.ca/about/academic-regulation-14-other-important-information
- http://web5.uottawa.ca/mcs-smc/academicintegrity/home.php

Cases of plagiarism will be dealt with according to the university regulations.

By submitting this assignment, you acknowledge:

1. **having read the academic regulations regarding academic fraud, and**
2. **understanding the consequences of plagiarism**