Patrick Charles prc9219
CMPS 450
Assignment 3 – Object Oriented Programming
Fall 15

Object Oriented Programming

I. Description of Assignment Goals

The goal of this assignment is to familiarize students with the object-oriented programming language paradigm. Through this assignment students will deeply learn how the OOP works and can be extended.

II. Language and platform

Written in Netbeans IDE v8.0.2
Product Version: NetBeans IDE 8.0.2 (Build 201411181905)
Updates: NetBeans IDE is updated to version NetBeans 8.0.2 Patch 2
Java: 1.8.0_66; Java HotSpot(TM) 64-Bit Server VM 25.66-b18
Runtime: Java(TM) SE Runtime Environment 1.8.0_66-b18
System: Windows 10 version 10.0 running on amd64; Cp1252; en_US (nb)

III. Description of Programs

Overview:

III.1.1 ArrayStack

Description:
Develop a new type of stack class called ArrayStack as part of Java Collection Framework (JCF). This implementation of the stack uses ArrayList for storing its elements and will have all restrictions defined for the stack structure including push, pop, and peek (see the top element of stack) as well as the Collection methods size, add, and iterator. Students are suggested to achieve this by adding a type under the AbstractCollection class in JCF.

III.1.2 ChainStack

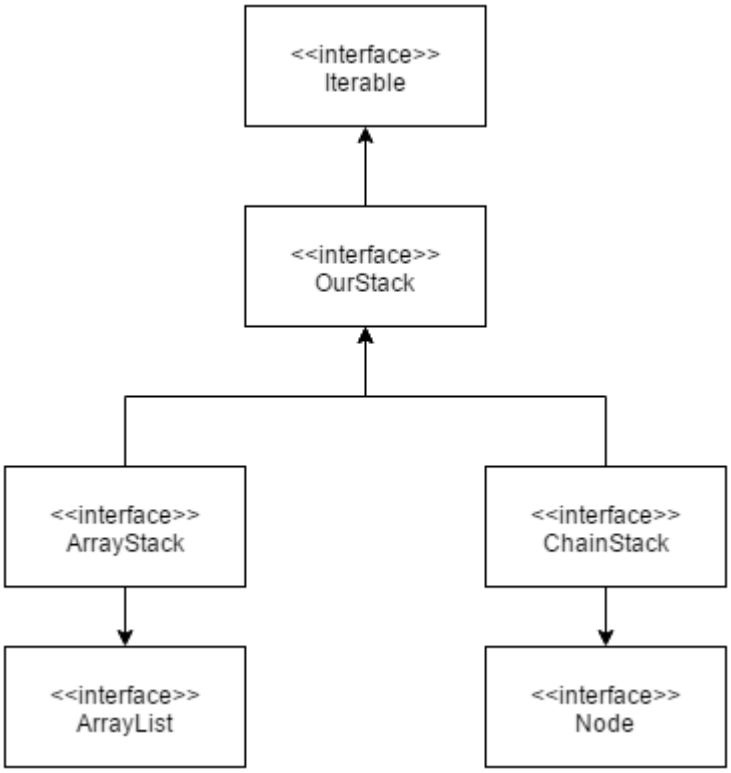
Description:
Develop an alternative solution for ArrayStack and call it ChainStack. It must be implemented in form of a singly linked sequence of nodes to hold the elements of the stack. It should have a single pointer called top that points to the first node of the stack. The structure of the stack node can be defined as an inner class within the ChainStack class. Remember that, in this case, a size

variable is needed to keep the number of elements exists in the stack at each point. Also, please note that for the iterator method a reference to top is required to know the beginning of the ChainStack.

III.1.3 Interface for ArrayStack and ChainStack

Description:
Implement an Interface (e.g., called OurStack) for the types implemented in part A and B. Then, update the implementation of A and B to implement interface OurStack. Draw the updated JCF hierarchy in the report and explain the essence and benefits of having this interface in the hierarchy.

III.1.3.1 Interface for ArrayStack and ChainStack Hierarchy



The essence of having a interface is because you have two different types of stacks, ArrayStack and ChainStack.

By using a interface with the same methods, you just need to change the implementations slightly of each and you can use either type to store data in a stack depending on what your needs may be.

With the single interface, all the standard stack methods are available to you, you just need to pick whether you want to store data in a linked list or array when creating your object.
Example:

OurStack arrayStack = new ArrayStack(); creates a new object that stores data in an Array structure
OurStack chainStack = new ChainStack(); creates an object that stores data in a linked list stricture

Using a OurStack object gives you access to all the standard methods of both types by using the same commands. An interface provides a standardization and simplifies the program for the users while providing different options in data structure.

III.1.4 Parenthesis Matching

Description:
A program needs to be developed to receive a programming statement (e.g., ((a + b) + (c + d));) as input and uses the stack structures defined in A or B to verify the correctness of parentheses in the given statement.

IV.1 Function descriptions

IV.1.1 ArrayStack

IV.1.1.1 ArrayStack Classes

ArrayStack:
Uses ArrayList class to store and manipulate data in a stack structure implementing standard stack functions such as peek(), pop(), and push()

private class ArrayStackIterator implements Iterator<T>:
implements Iterator for ArrayStack

ArrayStackDemo:
Main class used for testing ArrayStack class functionality

IV.1.1.2 ArrayStack Attributes

private ArrayList<T> arrayList;
used to store data of generic type

private int index;
index of iterator class used to determine position based on the size of the stack

IV.1.1.3 ArrayStack Functions

public class ArrayStack<T> extends ArrayList<T>

public ArrayStack():

constructor. Iniatilze stack to default size of 10

public void ArrayStack(int initialSize):
constructor. Iniatilze stack to user indicated size

public boolean isEmpty():
check if stack is empty

public void push(T element)
push data to top of stack

public T pop()
remove data at top of stack and resize stack

public T peek()
returns data at top of stack

public T peek(int x)
returns data at the user indicated index

public int size()
returns current size of stack

public void printStack():
prints out current contents of stack stored in ArrayList<T> private object

private class ArrayStackIterator implements Iterator<T>:

public ArrayStackIterator()
initialize Iterator index to top of stack

public boolean hasNext()
determines if current index is top of the stack. (index > 0 since index is top of the stack).

public T next()
returns next element in stack by decrementing index and retrieving using object.get() function of ArrayList object where data is stored.

public void remove()
throws UnsupportedOperationException error

IV.1.2 ChainStack

IV.1.2.1 ChainStack Classes

public class ChainStack<T> implements Iterable<T>:

private class Node:

IV.1.2.2 ChainStack Attributes

public class ChainStack<T> implements Iterable<T>:

private int size;
keep track of size of stack

private Node top;
reference to node object that is at the top of the stack

private class Node:

private T data;
used to store generic data type of node

private Node next;
reference to next node in stack from current one

IV.1.2.3 ChainStack Functions

public class ChainStack<T> implements Iterable<T>

public void ChainStack():
default constructor to create linked list stack structure

public boolean isEmpty():
checks it stack is empty

public int getSize():
returns the size of the stack

public T pop():
removes the data at the top of the data and returns the removed item

public T peek():
returns data at the top of the stack

public void printStack():
prints contents of stack using iterator

public ChainStack<T> push(T x):
takes pushed data and adds to it to the top node of stack and creates a new node and updates “next” link

public Iterator<T> iterator():
creates ListIterator object for traversal of stack

private class ListIterator implements Iterator<T>

ListIterator():
Constructor. Sets current node to top of stack

public boolean hasNext():
returns true/false if current node is top of stack

public void remove():
not implemented. Throws UnsupportedOperationException error

public T next():
if stack has another node, returns data of ‘next’ node in the stack

public void printStack():
prints contents of stack using Iterator

IV.1.3 Interface for ArrayStack and ChainStack

IV.1.1.1 Interface for ArrayStack and ChainStack Classes

public class ChainStack<T> implements OurStack<T>

private class Node

private class ListIterator implements Iterator<T>

IV.1.1.2 Interface for ArrayStack and ChainStack Attributes

public class ChainStack<T> implements OurStack<T>:

private int size;
stores current size of stack
private Node top;
top node of the stack. stores data of node and link to next node in stack

private class Node:

private T data;
stores data of generic type for this particular node

private Node next;
link to next node in list

private class ListIterator implements Iterator<T>:

private Node current;
stores current node. Used as a placeholder that to iterate through list.

IV.1.1.3 Interface for ArrayStack and ChainStack Functions

public void ChainStack():
default constructor to create linked list stack structure

public boolean isEmpty():
checks it stack is empty

public int getSize():
returns the size of the stack

public T pop():
removes the data at the top of the data and returns the removed item

public T peek():
returns data at the top of the stack

public void printStack():
prints contents of stack using iterator

public void push(T x):
takes pushed data and adds to it to the top node of stack using ChainStack<T> add(T x) function

public ChainStack<T> add(T x):
takes pushed data and adds to it to the top node of stack and creates a new node and updates “next” link

public Iterator<T> iterator():
creates ListIterator object for traversal of stack

private class ListIterator implements Iterator<T>

ListIterator():
Constructor. Sets current node to top of stack

public boolean hasNext():
returns true/false if current node is top of stack

public void remove():
not implemented. Throws UnsupportedOperationException error

public T next():
if stack has another node, returns data of 'next' node in the stack

public void printStack():
prints contents of stack using Iterator

IV.1.1.4 Interface for ArrayStack and ChainStack Interface

public interface OurStack<T>

public boolean isEmpty();
Return whether the stack is empty.

public void push(T x);
Push x onto the stack.

public T peek();
Return the object at the top of the stack.

public T pop();
Remove and return the object at the top of the stack.

public void printStack();
Print contents of stack

V.1.4 Parenthesis matching

V.1.4.1 Classes

public class ArrayStack<T> extends ArrayList<T>:
creates object to store phrase as string and parse into characters in a stack structure

V.1.4.2 Parenthesis matching Attributes

public ArrayList<T> arrayList;

stores characters from parsed string in array

V.1.4.3 Parenthesis matching Functions

public ArrayStack():
constructor creates stack

public void ArrayStack(int initialSize)
constructor creates stack of user inputted size

public boolean isEmpty():
returns true is stack is empty

public void push(T element)
adds element to top of stack

public T pop()
removes top element of stack

public T peek()
returns top element of stack

public T peek(int x)
returns element at user indicated index

public int size()
returns size of stack

public boolean CheckParentesis(String string)

V.1 Source code for Programs

V.1.1 ArrayStack

```
package arraystackdemo;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.EmptyStackException;
import java.util.NoSuchElementException;

public class ArrayStack<T> extends ArrayList<T>
{
   private ArrayList<T> arrayList;

   public ArrayStack()
```

```
{
    arrayList = new ArrayList(10);
}

public void  ArrayStack(int initialSize)
{
    arrayList = new ArrayList(initialSize);
}

@Override
public boolean isEmpty()
{
    return (arrayList.isEmpty());
}
public void push(T element)
{
    arrayList.add(element);
}

public T pop()
{
    if (!isEmpty())
        return arrayList.remove(size()-1);
    else
        throw new EmptyStackException();
}
public T peek()
{
 if (!isEmpty())
     return arrayList.get(size()-1);
 else
       throw new EmptyStackException();
}

public T peek(int x)
{
 int y = ((arrayList.size()-x)-1);

 if (y < 0)
     throw new EmptyStackException();
 else
      return arrayList.get(y);

}

@Override
```

```
public int size()
{
    return arrayList.size();
}

public void printStack()
{
   System.out.println(arrayList);
}

// Returns an Iterator to traverse the elements of this stack.
@Override
 public Iterator<T> iterator()
 {
       return new ArrayStackIterator();
 }

 private class ArrayStackIterator implements Iterator<T>
 {
    private int index;

    public ArrayStackIterator()
    {
       index = arrayList.size()- 1;
    }
    @Override
    public boolean hasNext()
    {
       return index >= 0;
    }

    @Override
    public T next()
    {
       if (!hasNext())
       {
          throw new NoSuchElementException();
       }
       T result = arrayList.get(index);
       index--;
       return result;
    }

    @Override
    public void remove()
    {
```

```
            throw new UnsupportedOperationException();
        }
    }
}
```

V.1.2 ChainStack

```
package ChainStackDemo;


import java.util.Iterator;
import java.util.NoSuchElementException;

public class ChainStack<T> implements Iterable<T>
{
    private int size;
    private Node top;

    private class Node
    {
        private T data;
        private Node next;
    }

public void ChainStack()
{
    top = null;
    size = 0;
}

public boolean isEmpty()
{
    return top == null;
}

public int getSize()
{
    return size;
}

public T pop()
{
    if (top == null)
        throw new NoSuchElementException();
    T data1 = top.data;
    top = top.next;
```

```
    size--;
    return data1;
}

public T peek()
{
    if (top == null)
        throw new NoSuchElementException();
    return top.data;

}

public ChainStack<T> push(T x)
    {
        Node current = top;
        top = new Node();
        top.data = x;
        top.next = current;
        size++;
        return this;
    }

public void printStack()
    {
        if (!isEmpty())
       {
            Iterator it = iterator();
            while (it.hasNext())
            System.out.println(it.next());
       }
    }

@Override
public Iterator<T> iterator()
{
    return new ListIterator();
}

// an iterator, doesn't implement remove() since it's optional
private class ListIterator implements Iterator<T>
{
    private Node current;

    ListIterator()
    {
        current = top;
```

```
    }

    @Override
    public boolean hasNext()
    {
        return current != null;
    }

    @Override
    public void remove()
    {
        throw new UnsupportedOperationException();
    }

    @Override
    public T next()
    {
        if(!hasNext()){
            throw new NoSuchElementException();}
        T item = current.data;
        current = current.next;
        return item;
    }
}
}
```

V.1.3 Interface for ArrayStack and ChainStack

V.1.3.1 Interface for ArrayStack and ChainStack – ArrayStack

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package interfacedemo;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.EmptyStackException;
import java.util.NoSuchElementException;


public class ArrayStack<T> implements OurStack<T>
{
   private ArrayList<T> arrayList;
```

```
public ArrayStack()
{
    arrayList = new ArrayList(20);
}

public void  ArrayStack(int initialSize)
{
    arrayList = new ArrayList(initialSize);
}

@Override
public boolean isEmpty()
{
    return (arrayList.isEmpty());
}

 @Override
 public void push(T x)
 {
     arrayList.add(x);
 }

@Override
public T pop()
{
    if (!isEmpty())
        return arrayList.remove(size()-1);
    else
        throw new EmptyStackException();
}
@Override
public T peek()
{
 if (!isEmpty())
     return arrayList.get(size()-1);
 else
       throw new EmptyStackException();
}


public T peek(int x)
{
 int y = ((arrayList.size()-x)-1);

 if (y < 0)
```

```
    throw new EmptyStackException();
 else
     return arrayList.get(y);

}

public int size()
{
    return arrayList.size();
}

@Override
public void printStack()
{
   System.out.println(arrayList);
}

// Returns an Iterator to traverse the elements of this stack.

 public Iterator<T> iterator()
 {
      return new ArrayStackIterator();
 }


 private class ArrayStackIterator implements Iterator<T>
 {
    private int index;

    public ArrayStackIterator()
    {
       index = arrayList.size()- 1;
    }
    @Override
    public boolean hasNext()
    {
       return index >= 0;
    }

    @Override
    public T next()
    {
       if (!hasNext())
       {
          throw new NoSuchElementException();
       }
```

```
            T result = arrayList.get(index);
            index--;
            return result;
        }

        @Override
        public void remove()
        {
            throw new UnsupportedOperationException();
        }
    }
}
```

V.1.3.1 Interface for ArrayStack and ChainStack – ChainStack

```
package interfacedemo;


import java.util.Iterator;
import java.util.NoSuchElementException;

public class ChainStack<T> implements OurStack<T>
{
    private int size;
    private Node top;

    private class Node
    {
        private T data;
        private Node next;
    }

    public void ChainStack()
    {
        top = null;
        size = 0;
    }

    public boolean isEmpty()
    {
        return top == null;
    }

    public int getSize()
    {
```

```
        return size;
    }

    @Override
    public T pop()
    {
        if (top == null)
            throw new NoSuchElementException();
        T data1 = top.data;
        top = top.next;
        size--;
        return data1;
}

    @Override
    public T peek()
{
    if (top == null)
        throw new NoSuchElementException();
    return top.data;

}
    @Override
    public void push(T x)
    {
        this.add(x);
    }

    @Override
    public void printStack()
    {
        if (!isEmpty())
        {
            Iterator it = iterator();
            while (it.hasNext())
            System.out.println(it.next());
        }
    }

    public ChainStack<T> add(T x)
    {
        Node current = top;
        top = new Node();
        top.data = x;
        top.next = current;
        size++;
```

```
        return this;
    }

    public Iterator<T> iterator()
    {
        return new ListIterator();
    }

    // an iterator, doesn't implement remove() since it's optional
    private class ListIterator implements Iterator<T>
    {
        private Node current;

    ListIterator()
    {
        current = top;
    }

    @Override
    public boolean hasNext()
    {
        return current != null;
    }

    @Override
    public void remove()
    {
        throw new UnsupportedOperationException();
    }

    @Override
    public T next()
    {
        if(!hasNext()){
            throw new NoSuchElementException();}
        T item = current.data;
        current = current.next;
        return item;
    }
}
}
```

V.1.3.1 Interface for ArrayStack and ChainStack – OurStack

```
package interfacedemo;
```

```
public interface OurStack<T>
{
  public boolean isEmpty();  // Return whether the stack is empty.

  public void push(T x);  // Push x onto the stack.

  public T peek();    // Return the object at the top of the stack.

  public T pop(); // Remove and return the object at the top of the stack.

  public void printStack(); //Print current contents of the stack
}
```

V.1.4 Parenthesis matching

```
package parenthesisdemo;

import java.util.ArrayList;
import java.util.EmptyStackException;


public class ArrayStack<T> extends ArrayList<T>
{
   public ArrayList<T> arrayList;

   public ArrayStack()
   {
      arrayList = new ArrayList(10);
   }

   public void  ArrayStack(int initialSize)
   {
      arrayList = new ArrayList(initialSize);
   }

   @Override
   public boolean isEmpty()
   {
      return (arrayList.isEmpty());
   }
   public void push(T element)
   {
      arrayList.add(element);
   }
```

```
    public T pop()
    {
        if (!isEmpty())
            return arrayList.remove(size()-1);
        else
            throw new EmptyStackException();
    }
    public T peek()
    {
     if (!isEmpty())
         return arrayList.get(size()-1);
     else
           throw new EmptyStackException();
    }

    public T peek(int x)
    {
     int y = ((arrayList.size()-x)-1);

     if (y < 0)
         throw new EmptyStackException();
     else
          return arrayList.get(y);

    }

    @Override
    public int size()
    {
        return arrayList.size();
    }

public boolean CheckParentesis(String string)
{
    if (string.isEmpty())
        return true;

    ArrayStack<Character> stack = new ArrayStack();

    for (int i = 0; i < string.length(); i++)
    {
        char current = string.charAt(i);

        if (current == '{' || current == '(' || current == '[')
        {
            stack.push(current);
```

```
        }


        if (current == '}' || current == ')' || current == ']')
        {
            if (stack.isEmpty())
                return false;

            char last = stack.peek();

            if (current == '}' && last == '{' || current == ')' && last
                == '(' || current == ']' && last == '[')
                stack.pop();
            else
                return false;
        }
    }
    return stack.isEmpty();
}
}
```

VI.1 Test Case for Programs

VI.1.1 ArrayStack

Test Code:

```
        ArrayStack a = new ArrayStack();
        System.out.println("push (10,20,30,40,50,60) to stack");
        a.push(10);
        a.push(20);
        a.push(30);
        a.push(40);
        a.push(50);
        a.push(60);

         System.out.println("print stack");
         a.printStack();

         System.out.println("pop stack : " + a.pop());

         System.out.println("get size of stack : " + a.size());

         System.out.println("print stack");
         a.printStack();
```

```
System.out.println("is stack empty : " +a.isEmpty());

System.out.println("peek stack : " + a.peek());


System.out.println("print array stack using iterator");
if (!a.isEmpty())
{
    Iterator iterator = a.iterator();
    while (iterator.hasNext())
    {
    System.out.println(iterator.next());
    }
}
```

Output:

```
run:
push (10,20,30,40,50,60) to stack
print stack
[10, 20, 30, 40, 50, 60]
pop stack : 60
get size of stack : 5
print stack
[10, 20, 30, 40, 50]
is stack empty : false
peek stack : 50
print array stack using iterator
50
40
30
20
10
BUILD SUCCESSFUL (total time: 0 seconds)
```

VI.1.2 ChainStack

Test code:

```
ChainStack a = new ChainStack();
System.out.println("push (1,2,3,a,b) to stack");
a.push(1);
a.push(2);
a.push(3);
a.push("a");
a.push("b");
```

```
System.out.println("print stack");
a.printStack();

System.out.println("pop stack : " + a.pop());

System.out.println("get size of stack : " + a.getSize());

System.out.println("print stack");
a.printStack();

System.out.println("is stack empty : " +a.isEmpty());

System.out.println("peek stack : " + a.peek());
```

Output:

```
run:
push (1,2,3,a,b) to stack
print stack
b
a
3
2
1
pop stack : b
get size of stack : 4
print stack
a
3
2
1
is stack empty : false
peek stack : a
BUILD SUCCESSFUL (total time: 0 seconds)
```

VI.1.3 Interface for ArrayStack and ChainStack

Test Code:

```
package interfacedemo;

public class InterfaceDemo {
```

```
public static void main(String[] args)

{
    OurStack arrayStack;
    OurStack chainStack;

    arrayStack = new ArrayStack();
    chainStack = new ChainStack();
    System.out.println("ArrayStack interface demo : ");

    System.out.println("push (10,20,30,40,50,60) to stack");
    arrayStack.push(10);
    arrayStack.push(20);
    arrayStack.push(30);
    arrayStack.push(40);
    arrayStack.push(50);
    arrayStack.push(60);

    System.out.println("print stack");
    arrayStack.printStack();

    System.out.println("pop stack : " + arrayStack.pop());

    System.out.println("get size of stack : " + arrayStack.size());

    System.out.println("print stack");
    arrayStack.printStack();

    System.out.println("is stack empty : " +arrayStack.isEmpty());

    System.out.println("peek stack : " + arrayStack.peek());

    System.out.println("ChainStack interface demo : ");

    System.out.println("push (1,2,3,a,b) to stack");
    chainStack.push(1);
    chainStack.push(2);
    chainStack.push(3);
    chainStack.push("a");
    chainStack.push("b");

    System.out.println("print stack");
    chainStack.printStack();

    System.out.println("pop stack : " + chainStack.pop());
```

```
        System.out.println("get size of stack : " + chainStack.size());

        System.out.println("print stack");
        chainStack.printStack();

        System.out.println("is stack empty : " +chainStack.isEmpty());

        System.out.println("peek stack : " + chainStack.peek());
    }

}
```

Output:

```
run:
ArrayStack interface demo :
push (10,20,30,40,50,60) to stack
print stack
[10, 20, 30, 40, 50, 60]
pop stack : 60
get size of stack : 5
print stack
[10, 20, 30, 40, 50]
is stack empty : false
peek stack : 50
ChainStack interface demo :
push (1,2,3,a,b) to stack
print stack
b
a
3
2
1
pop stack : b
get size of stack : 4
print stack
a
3
2
1
is stack empty : false
peek stack : a
BUILD SUCCESSFUL (total time: 0 seconds)
```

VI.1.4 Parenthesis matching

Test Code:

```
package parenthesisdemo;

public class ParenthesisDemo {


    public static void main(String[] args)
    {
        ArrayStack a = new ArrayStack();
        String s = "(((((a + b) + (c + d)))"; //incorrect
        String t = "a + (c + d)";  // correct
        String u = "((a + b) * (c - d))"; //correct
        String v = "(a + b))"; //incorrect

        a.CheckParenthesis(s);
        a.CheckParenthesis(t);
        a.CheckParenthesis(u);
        a.CheckParenthesis(v);

    }
}
```

Output:

```
run:
parenthesis [ (((((a + b) + (c + d))) ] is incorrect
parenthesis [ a + (c + d) ] is correct
parenthesis [ ((a + b) * (c - d)) ] is correct
parenthesis [ ((a + b) ] is incorrect
BUILD SUCCESSFUL (total time: 0 seconds)
```