CHAPTER 7 : STACKS

THE STACK ADT IS ESSENTIALLY A LIST IN WHICH ALL INSERTIONS AND DELETIONS ARE DONE AT THE SAME END. THUS THE LAST ITEM INSERTED WILL BE THE FIRST ITEM DELETED OR TAKEN OUT. WE CALL THIS PROPERTY LAST IN, FIRST OUT OR LIFO.

THE BOOK DISCUSSES STACKS IN WHICH THE ELEMENTS ARE Objects, WHEREAS WE WILL SIMPLIFY THIS TO A STACK OF INTEGERS.

THE IntegerStack OPERATIONS ARE:

Public boolean isEmpty()
Pre: none
Post: RETURNS TRUE IF STACK IS EMPTY, FALSE OTHERWISE

Public void push(int item)
Pre: none
Post: PLACES NEW item ON STACK

Public int pop()
Pre: ! isEmpty()
Post: DELETES & RETURNS TOP item ON STACK.

Public int peek()
Pre: ! isEmpty()
Post: returns item ON TOP OF STACK

Public void popAll()
Pre: !isEmpty()
Post: isEmpty()

NOTE: THE BOOK DOES NOT INCLUDE !isEmpty() AS A PRECONDITION TO pop(), peek(), AND popAll().

CERTAIN IMPLEMENTATIONS MAY REQUIRE ADDITIONAL PRE-CONDITIONS. FOR INSTANCE AN ARRAY OF FIXED SIZE WOULD REQUIRE THAT WE CHECK IF THE ARRAY IS FULL BEFORE DOING push().

WE CONSIDER THREE IMPLEMENTATIONS:

- ARRAY BASED WITH ARRAY DOUBLING.
- LINKED LIST BASED.
- BASED ON IntegerList ADT.

READ EXAMPLE
EVALUATING ALGEBRAIC EXPRESSIONS IN POSTFIX AND INFIX FORMS.

```
// IntegerStackInterface.java
// interface for the IntegerStack ADT

public interface IntegerStackInterface{

   // isEmpty
   // pre: none
   // post: returns true if this IntgerStack is empty, false otherwise
   public boolean isEmpty();

   // push
   // pushes x onto top of this IntegerStack
   // pre: none
   // post: !isEmpty()
   public void push(int x);

   // pop
   // deletes and returns item from top of Stack
   // pre: !isEmpty()
   // post: this Stack will have one fewer element
   public int pop() throws StackEmptyException;

   // peek
   // pre: !isEmpty()
   // post: returns item on top of Stack
   public int peek() throws StackEmptyException;

   // popAll
   // pre: !isEmpty()
   // post: isEmpty()
   public void popAll() throws StackEmptyException;
}
```

```
// IntegerStack.java
// Array based implementation of IntegerStack ADT (with array doubling)

public class IntegerStack implements IntegerStackInterface{
   private static final int INITIAL_SIZE = 1;
   private int physicalSize;  // current length of underlying array
   private int[] item;        // array of IntegerStack items
   private int numItems;      // number of items in this IntegerStack
   private int top;           // index of top element of Stack in array

   // doubleItemArray
   // doubles the physical size of the underlying array item[]
   private void doubleItemArray(){
      physicalSize *=2;
      int[] newArray = new int[physicalSize];
      for(int i=0; i<numItems; i++) newArray[i] = item[i];
      item = newArray;
   }

   // IntegerStack
   // default constructor for the IntegerStack class
   public IntegerStack(){
      physicalSize = INITIAL_SIZE;
      item = new int[physicalSize];
      numItems = 0;
      top = -1;       // indicates empty Stack
   }

   // isEmpty
   // pre: none
   // post: returns true if this IntgerStack is empty, false otherwise
   public boolean isEmpty(){
      return(numItems == 0);   // could also check top<0
   }

   // push
   // pushes x onto top of this IntegerStack
   // pre: none
   // post: !isEmpty()
   public void push(int x){
      if( numItems == physicalSize ) doubleItemArray();
      numItems++;
      top++;
      item[top] = x;
   }

   // pop
   // deletes and returns item from top of Stack
   // pre: !isEmpty()
   // post: this Stack will have one fewer element
   public int pop() throws StackEmptyException{
      if( numItems==0 ){  // could also check top < 0
         throw new StackEmptyException("cannot pop() empty stack");
      }
      int returnValue = item[top];
      top--;
      numItems--;
      return returnValue;
   }
```

```
    // peek
    // pre: !isEmpty()
    // post: returns item on top of Stack
    public int peek() throws StackEmptyException {
       if( numItems==0 ){  // could also check top<0
          throw new StackEmptyException("cannot peek() empty stack");
       }
       return item[top];
    }

    // popAll
    // pre: !isEmpty()
    // post: isEmpty()
    public void popAll() throws StackEmptyException{
       if( numItems==0 ){  // could also check top < 0
          throw new StackEmptyException("cannot popAll() empty stack");
       }
       numItems = 0;
       top = -1;
    }

    // toString
    // overrides Object's toString() method
    public String toString(){
       String s = "";

       for(int i=numItems-1; i>=0; i--){
          s += item[i] + " ";
       }
       return s;
    }

    // equals
    // overrides Object's equals
    public boolean equals(Object rhs){
       IntegerStack R = null;
       boolean eq = false;

       if(rhs instanceof IntegerStack){
          R = (IntegerStack)rhs;
          eq = ( this.numItems == R.numItems );
          for(int i=0; eq && i<numItems; i++){
             eq = ( this.item[i] == R.item[i] );
          }
       }
       return eq;
    }

}
```

```
// IntegerStackTest.java
// Test Client for the IntegerStack class

public class IntegerStackTest{

   public static void main(String[] args){
      IntegerStack A = new IntegerStack();

      A.push(5); A.push(3); A.push(9); A.push(7); A.push(8);
      System.out.println(A);
      System.out.println(A.peek());

      A.pop(); A.pop(); A.pop();
      System.out.println(A.peek());
      System.out.println(A);

      IntegerStack B = new IntegerStack();
      System.out.println(A.isEmpty());
      System.out.println(B.isEmpty());

      B.push(5); B.push(3);
      System.out.println(A.equals(B));

      A.push(12);
      B.push(13);
      System.out.println(A);
      System.out.println(B);
      System.out.println(A.equals(B));

      A.popAll();
      System.out.println(A);
      System.out.println(A.isEmpty());
   }
}
```

StackEmptyException.java

```
// StackEmptyException.java

public class StackEmptyException extends RuntimeException{
   public StackEmptyException(String s){
      super(s);
   }
}
```

```
// IntegerStack.java
// Linked List implementation of IntegerStack ADT

public class IntegerStack implements IntegerStackInterface{

   private class Node{
      int item;
      Node next;
      Node(int item){
         this.item = item;
         this.next = null;
      }
   }

   private Node top;       // reference to the top Node in the stack
   private int numItems;       // number of items in this IntegerStack

   // IntegerStack
   // default constructor for the IntegerStack class
   public IntegerStack(){
      top = null;
      numItems = 0;
   }

   // isEmpty
   // pre: none
   // post: returns true if this IntgerStack is empty, false otherwise
   public boolean isEmpty(){
      return(numItems == 0);    // could also check top<0
   }

   // push
   // pushes x onto top of this IntegerStack
   // pre: none
   // post: !isEmpty()
   public void push(int x){
      Node N = new Node(x);
      N.next = top;
      top = N;
      numItems++;
   }

   // pop
   // deletes and returns item from top of Stack
   // pre: !isEmpty()
   // post: this Stack will have one fewer element
   public int pop() throws StackEmptyException{
      if( numItems==0 ){  // could also check top==null
         throw new StackEmptyException("cannot pop() empty stack");
      }
      int returnValue = top.item;
      top = top.next;
      numItems--;
      return returnValue;
   }

   // peek
   // pre: !isEmpty()
   // post: returns item on top of Stack
   public int peek() throws StackEmptyException {
      if( numItems==0 ){  // could also check top==null
         throw new StackEmptyException("cannot peek() empty stack");
      }
      return top.item;
   }
```

```
    // popAll
    // pre: !isEmpty()
    // post: isEmpty()
    public void popAll() throws StackEmptyException{
       if( numItems==0 ){  // could also check top==null
          throw new StackEmptyException("cannot popAll() empty stack");
       }
       top = null;
       numItems = 0;
    }

    // toString
    // overrides Object's toString() method
    public String toString(){
       String s = "";
       for(Node N = top; N!=null; N=N.next){
          s = s+String.valueOf(N.item)+" ";
       }
       return s;
    }

    // equals
    // overrides Object's equals
    public boolean equals(Object rhs){
       if(rhs instanceof IntegerStack){
          boolean equal = (numItems==((IntegerStack)rhs).numItems);
          Node N = top;
          Node M = ((IntegerStack)rhs).top;
          while(equal && N!=null){
             equal = (N.item==M.item);
             N = N.next;
             M = M.next;
          }
          return equal;
       }else{
          return false;
       }
    }

}
```

```
// IntegerStack.java
// IntegerStack ADT based on the IntegerList ADT

public class IntegerStack implements IntegerStackInterface{

   private IntegerList list;  // list of items in stack

   // IntegerStack
   // default constructor for the IntegerStack class
   public IntegerStack(){
      list = new IntegerList();
   }

   // isEmpty
   // pre: none
   // post: returns true if this IntgerStack is empty, false otherwise
   public boolean isEmpty(){
      return(list.isEmpty());
   }

   // push
   // pushes x onto top of this IntegerStack
   // pre: none
   // post: !isEmpty()
   public void push(int x){
      list.add(1, x);
   }

   // pop
   // deletes and returns item from top of Stack
   // pre: !isEmpty()
   // post: this Stack will have one fewer element
   public int pop() throws StackEmptyException{
      if( list.size()==0 ){
         throw new StackEmptyException("cannot pop() empty stack");
      }
      int returnValue = list.get(1);
      list.remove(1);
      return returnValue;
   }

   // peek
   // pre: !isEmpty()
   // post: returns item on top of Stack
   public int peek() throws StackEmptyException {
      if( list.size()==0 ){
         throw new StackEmptyException("cannot peek() empty stack");
      }
      return list.get(1);
   }

   // popAll
   // pre: !isEmpty()
   // post: isEmpty()
   public void popAll() throws StackEmptyException{
      if( list.size()==0 ){
         throw new StackEmptyException("cannot popAll() empty stack");
      }
      list.removeAll();
   }

   // toString
   // overrides Object's toString() method
   public String toString() { return list.toString(); }
```

```
    // equals
    // overrides Object's equals
    public boolean equals(Object rhs){
       IntegerStack R = null;
       Boolean eq = false;

       if(rhs instanceof IntegerStack){
          R = (IntegerStack)rhs;
          eq = ( this.list.equals(R.list) );
       }
       return eq;
    }
}
```