Exercise 2

```
class Hallo {

  public static void main (String[] args) {
    switch (args.length) {

      case 0: System.out.println("Who is there?"); break;

      default: System.out.print("Hallo ");

               for (int i = 0; i < args.length; i++) {
                    if (i == args.length-2) System.out.print(args[i] + " and ");

                   else if (i == args.length-1) System.out.println(args[i] + "!");

                   else System.out.print(args[i] + ", ");
               }
               break;
    }
  }
}
```

Exercise 3

```
public class Matrix{

  public static void main(String[] args){
    double[][] matrix1 = {{1.0, 2.0, 3.0, 4.0, 5.0}, {1.0, 2.0, 3.0, 4.0, 5.0}, {1.0,
2.0, 3.0, 4.0, 5.0}, {1.0, 2.0, 3.0, 4.0, 5.0}};

    double[][] matrix2 = {{1.0, 2.0, 3.0, 4.0}, {1.0, 2.0, 3.0, 4.0}, {1.0, 2.0, 3.0,
4.0}, {1.0, 2.0, 3.0, 4.0}, {1.0, 2.0, 3.0, 4.0}};

    int rows = matrix1.length;
    int cols = matrix2[0].length;
    double[][] product = new double[rows][cols];

    for(int i = 0; i < rows; i++)
     for(int j = 0; j < cols; j++)
      for(int k = 0; k < matrix1[0].length; k++)        //  or (k <matrix2.length)
        product[i][j] += matrix1[i][k]*matrix2[k][j];
```

Exercise 3 (continuation)

```
for(int i = 0; i < rows; i++) {
     for(int j = 0; j < cols; j++){
      System.out.print(product[i][j]+"\t");
     }
     System.out.println();
    }
  }
}
```

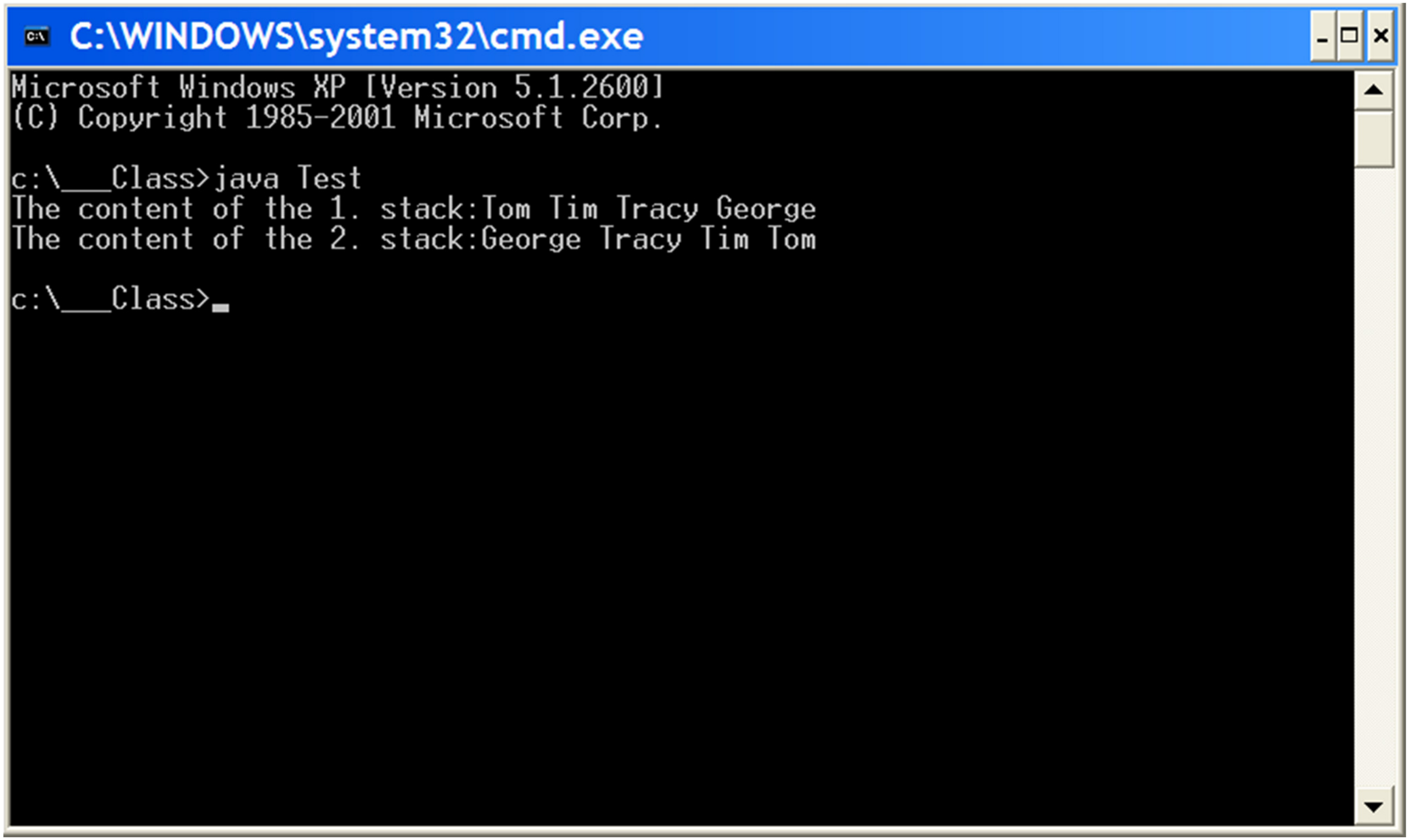
Exercise 4 – class Stack

```
public class Stack {

  protected String[] arrayStack;
  protected int top;
              //denotes the next idle place

  public Stack(int size) {
    this.arrayStack = new String[size];
    top = 0;
  }

  public void push(String s) {
    if (top == arrayStack.length) return;
    arrayStack[top++] = s;
  }

  public String pop() {
    if (top == 0) return "Error: Empty Stack";
    return arrayStack[--top];
  }

  public boolean isEmpty() {
    if (top == 0) return true;
    return false;
  }

  public String toString() {
    StringBuilder sb = new StringBuilder();
    for (int i=0; i < top; i++) {
       sb.append(arrayStack[i]+" ");
    }
    return sb.toString();
  }
}
```

Exercise 4 – class Test

```
public class Test {
  public static void main(String[] args) {
    Stack stack1 = new Stack(10);
    Stack stack2 = new Stack(10);

    stack1.push("Tom");
    stack1.push("Tim");
    stack1.push("Tracy");
    stack1.push("George");

    System.out.println("The content of the 1. stack:" + stack1);

    while (!stack1.isEmpty()) {
      stack2.push(stack1.pop());
    }

    System.out.println("The content of the 2. stack:" + stack2);

  }
}
```

Exercise 4 – class Output



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\___Class>java Test
The content of the 1. stack:Tom Tim Tracy George
The content of the 2. stack:George Tracy Tim Tom

c:\___Class>_
```

Exercise 5 – class *DebugStack*

```
public class DebugStack extends Stack {
  private String name;

  public DebugStack(int size, String name) {
    super(size);
    this.name = name;
    System.out.println(“DebugStack “+name+” is initialized.”);
  }
```

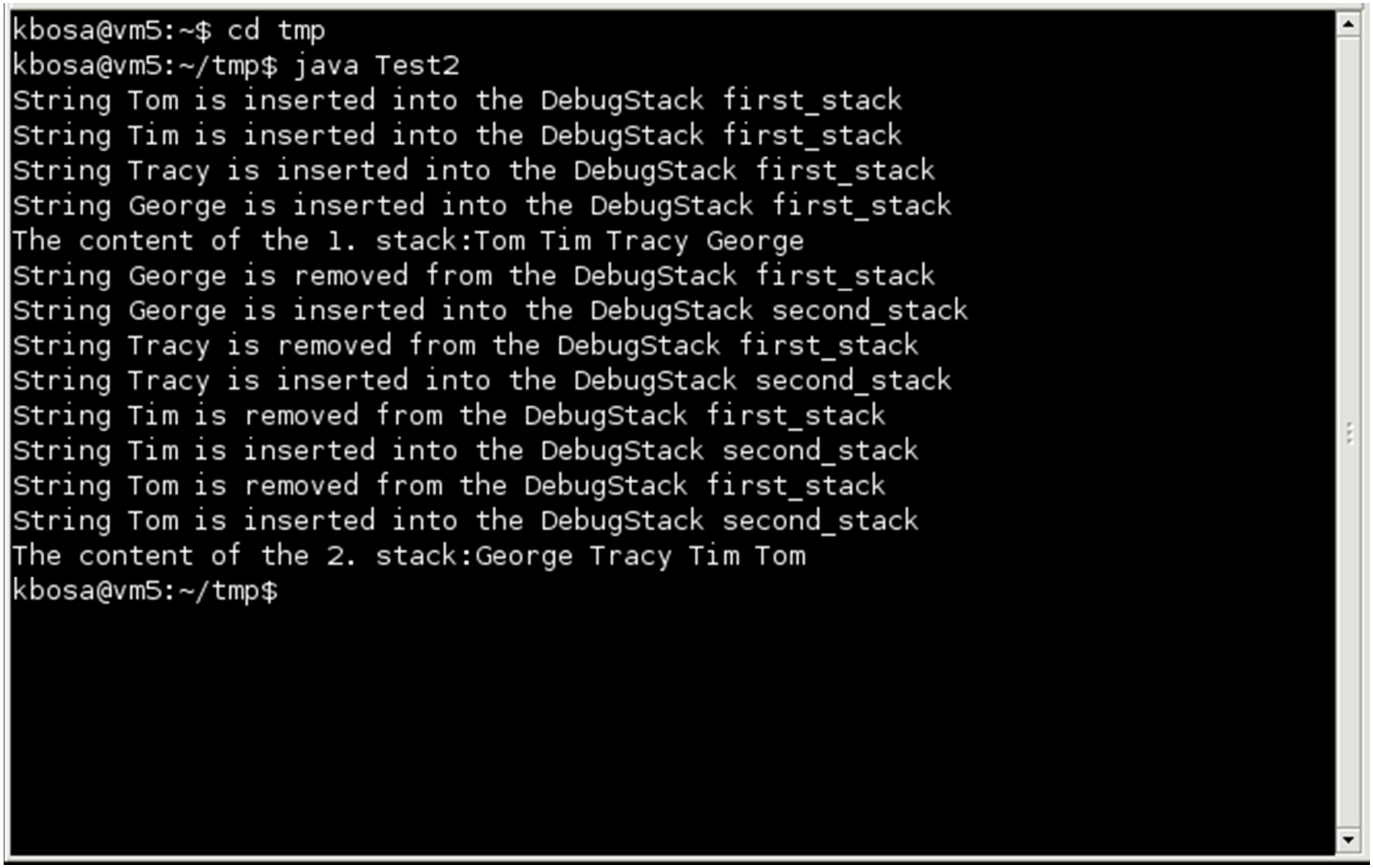
Exercise 5 – class *DebugStack* (cont.)

```
public void push(String s) {
  if (top == arrayStack.length) {
   System.out.println("Error: DebugStack " + name + " is full");
   return;
  }
  System.out.println("String " + s + " is inserted into the DebugStack " + name);
  arrayStack[top++] = s;
 }

 public String pop() {
  if (top == 0) {
   System.out.println("Error: DebugStack " + name + " is empty");
   return "Error: Empty Stack";
  }
  System.out.println("String " + arrayStack[--top] + " is removed from the DebugStack
" + name);
  return arrayStack[top];
 }

}
```

Exercise 5 – class *Test2*

```
public class Test2 {
  public static void main(String[] args) {
    DebugStack stack1 = new DebugStack(10, "first_stack");
    DebugStack stack2 = new DebugStack(10, "second_stack");

    stack1.push("Tom");
    stack1.push("Tim");
    stack1.push("Tracy");
    stack1.push("George");

    System.out.println("The content of the 1. stack:" + stack1);

    while (!stack1.isEmpty()) {
      stack2.push(stack1.pop());
    }

    System.out.println("The content of the 2. stack:" + stack2);

  }
}
```

Exercise 5 - Output



```
kbosa@vm5:~$ cd tmp
kbosa@vm5:~/tmp$ java Test2
String Tom is inserted into the DebugStack first_stack
String Tim is inserted into the DebugStack first_stack
String Tracy is inserted into the DebugStack first_stack
String George is inserted into the DebugStack first_stack
The content of the 1. stack:Tom Tim Tracy George
String George is removed from the DebugStack first_stack
String George is inserted into the DebugStack second_stack
String Tracy is removed from the DebugStack first_stack
String Tracy is inserted into the DebugStack second_stack
String Tim is removed from the DebugStack first_stack
String Tim is inserted into the DebugStack second_stack
String Tom is removed from the DebugStack first_stack
String Tom is inserted into the DebugStack second_stack
The content of the 2. stack:George Tracy Tim Tom
kbosa@vm5:~/tmp$
```

Exercise 6 - EmpyStackException

```
package stack;

public class EmptyStackException extends RuntimeException {
  public EmptyStackException() {
    super();
  }

  public EmptyStackException(String detail) {
    super(detail);
  }

}
```

Exercise 6 - FullStackException

```
package stack;

public class FullStackException extends RuntimeException {
  public FullStackException() {
    super();
  }

  public FullStackException(String detail) {
    super(detail);
  }

}
```

Exercise 6 – interface *Stack*

```
package stack;

public interface Stack {
  void clear();
  void exch() throws EmptyStackException;
  String peek() throws EmptyStackException;
  void push(String s) throws FullStackException;
  String pop() throws EmptyStackException;
  boolean isEmpty();
  String toString();
}
```

Exercise 6 – abstract class AbstractStack

Karoly.Bosa@jku.at

```
package stack;

public abstract class AbstractStack implements Stack {

   public abstract void push(String s) throws FullStackException;
   public abstract String pop() throws EmptyStackException;
   public abstract boolean isEmpty();
   public abstract String toString();


   …
}
```

Exercise 6 – abstract class AbstractStack

```
public void clear() {
  while(!isEmpty()) {
    pop();
  }
}

public void exch() throws EmptyStackException {
  String tmp1 = pop();
  String tmp2 = pop();

  push(tmp1);
  push(tmp2);
}

public String peek() throws EmptyStackException {
  String tmp = pop();
  push(tmp);
  return tmp;
}
```

Exercise 6 – class BoundedStack

```
package stack;

public class BoundedStack extends AbstractStack {

  protected String[] arrayStack;
  protected int top; //denotes the next idle place

  public BoundedStack(int size) {
    this.arrayStack = new String[size];
    top = 0;
  }

  public void push(String s) throws FullStackException {
    if (top == arrayStack.length) throw new FullStackException();
    arrayStack[top++] = s;
  }

  public String pop() throws EmptyStackException {
    if (top == 0) throw new EmptyStackException();
    return arrayStack[--top];
  }
```

Exercise 6 - BoundedStack

```
public boolean isEmpty() {
    if (top == 0) return true;
    return false;
  }

  public String toString() {
    StringBuilder sb = new StringBuilder();
    for (int i=0; i < top; i++) {
       sb.append(arrayStack[i]+" ");
    }
    return sb.toString();
  }

}
```

Methods isEmpty() and toString() are the same as before.

Exercise 6 – class Test3

```
package test;

import stack.*;

public class Test3 {
  public static void copy(Stack s1, Stack s2) throws FullStackException {
    while (!s1.isEmpty()) {
      s2.push(s1.pop());
    }
  }

 public static void main(String[] args) {
    …
 }
}
```

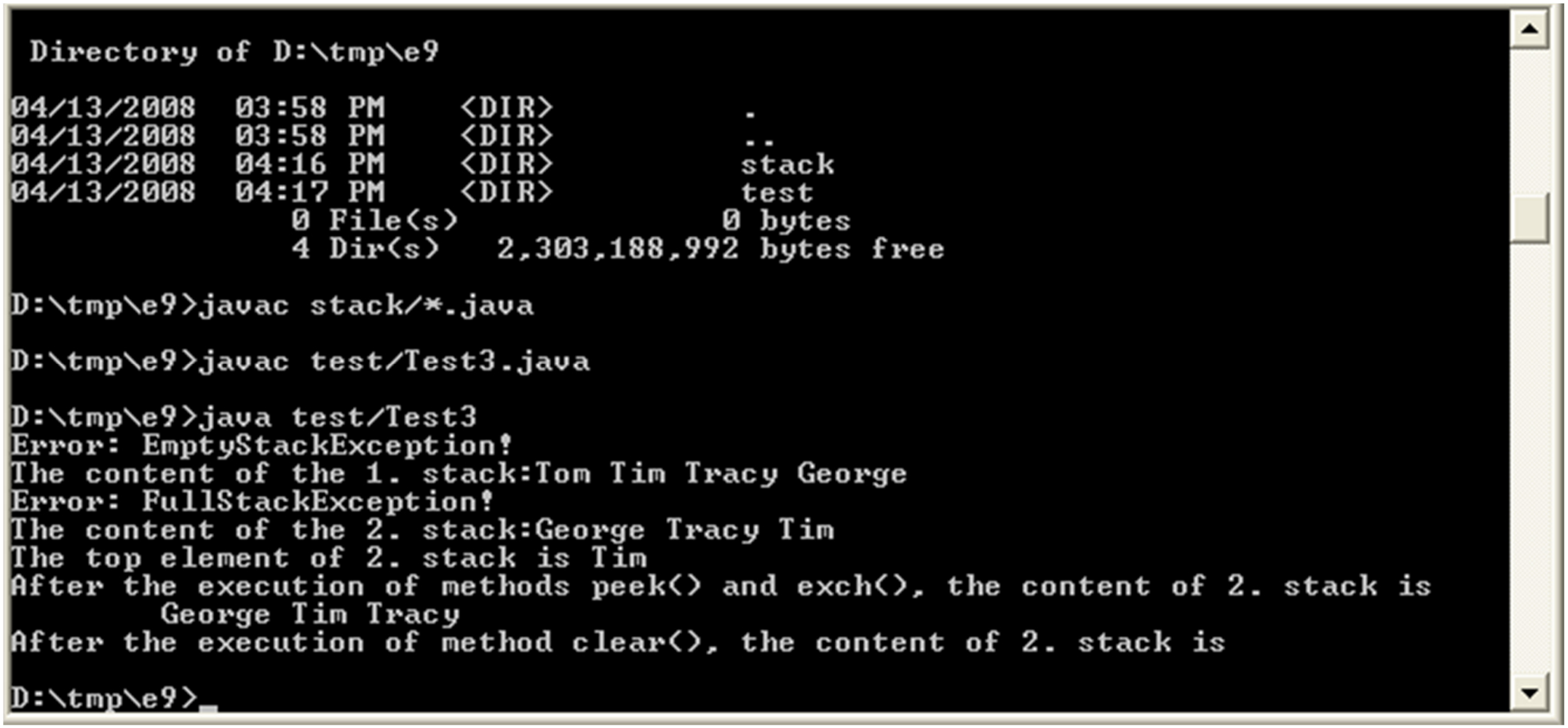
Exercise 6 – Test3

```
public static void main(String[] args) {
    Stack stack1 = new BoundedStack(10);
    Stack stack2 = new BoundedStack(3);

    try {
       stack1.pop();
    } catch(EmptyStackException e) {
       System.out.println("Error: EmptyStackException!");
    }

    try {
      stack1.push("Tom");
      stack1.push("Tim");
      stack1.push("Tracy");
      stack1.push("George");
      System.out.println("The content of the 1. stack:" + stack1);
      copy (stack1, stack2);
    } catch(FullStackException e) {
       System.out.println("Error: FullStackException!");
    }
```

Exercise 6 –Test3

```
finally {
      System.out.println("The content of the 2. stack:" + stack2);
      System.out.println("The top element of 2. stack is "+ stack2.peek());
      stack2.exch();
      System.out.println("After the execution of methods peek() and exch(), the 
content of 2. stack is \n\t"+stack2);
      stack2.clear();
      System.out.println("After the execution of method clear(), the content of 2. 
stack is "+stack2);
    }
  }
```

Exercise 6 – Output of Test3



```
 Directory of D:\tmp\e9

04/13/2008  03:58 PM    <DIR>          .
04/13/2008  03:58 PM    <DIR>          ..
04/13/2008  04:16 PM    <DIR>          stack
04/13/2008  04:17 PM    <DIR>          test
               0 File(s)              0 bytes
               4 Dir(s)   2,303,188,992 bytes free

D:\tmp\e9>javac stack/*.java

D:\tmp\e9>javac test/Test3.java

D:\tmp\e9>java test/Test3
Error: EmptyStackException!
The content of the 1. stack:Tom Tim Tracy George
Error: FullStackException!
The content of the 2. stack:George Tracy Tim
The top element of 2. stack is Tim
After the execution of methods peek() and exch(), the content of 2. stack is
        George Tim Tracy
After the execution of method clear(), the content of 2. stack is

D:\tmp\e9>_
```

Alternative manner for compilation from test directory:

javac –cp .. Test3.java

Alternative manner for execution from test directory:

java –cp .. Test/Test3

Exercise 6 – class DebugStack

```
package stack;

public class DebugStack {

  private String name;
  private Stack delegate;

  public DebugStack(String name, Stack delegate) {
    this.name = name;
    this.delegate = delegate;
  }

  public boolean isEmpty() {
    return delegate.isEmpty();
  }

  public String toString() {
    return delegate.toString();
  }

}
```

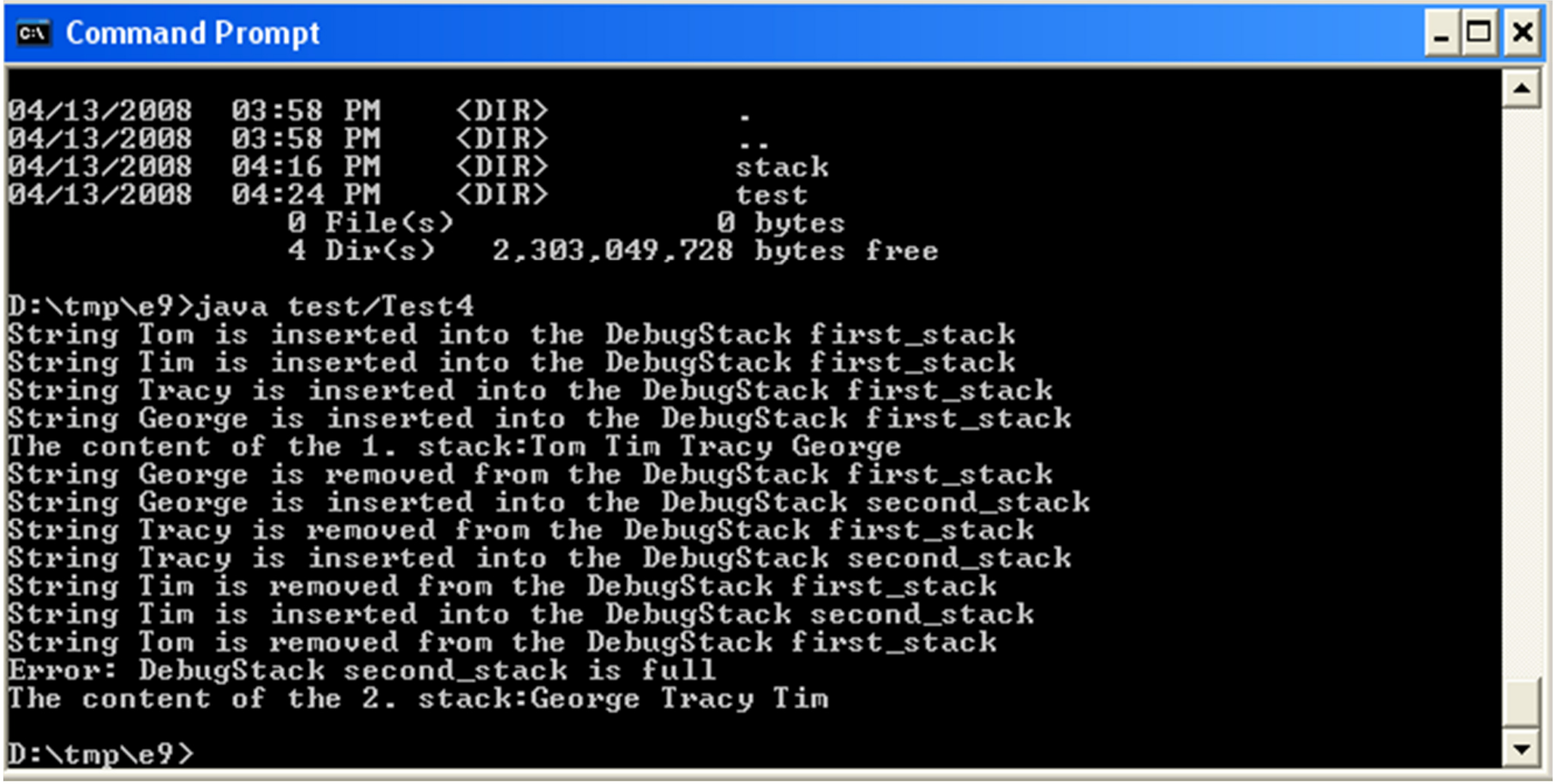
Exercise 6 – class DebugStack

```
 public void push(String s) {
   try {
     delegate.push(s);
     System.out.println("String " + s + " is inserted into the DebugStack " +
name);
   } catch (FullStackException e) {
     System.out.println("Error: DebugStack " + name + " is full");
   }
  }

  public String pop() {
   try {
     String tmp = delegate.pop();
     System.out.println("String " + tmp + " is removed from the DebugStack " +
name);
     return tmp;
   } catch (EmptyStackException e) {
     System.out.println("Error: DebugStack " + name + " is empty");
     return "Error: Empty Stack";
   }
  }
```

Exercise 6 - class Test4

```
package test;
import stack.*;
public class Test4 {
  public static void main(String[] args) {
    Stack s1 = new BoundedStack(10);
    Stack s2 = new BoundedStack(3);
    DebugStack stack1 = new DebugStack("first_stack", s1);
    DebugStack stack2 = new DebugStack("second_stack", s2);

    stack1.push("Tom");
    stack1.push("Tim");
    stack1.push("Tracy");
    stack1.push("George");
    System.out.println("The content of the 1. stack:" + stack1);

    while (!stack1.isEmpty()) {
      stack2.push(stack1.pop());
    }
    System.out.println("The content of the 2. stack:" + stack2);
  }
}
```

Exercise 6 – Output of Test4

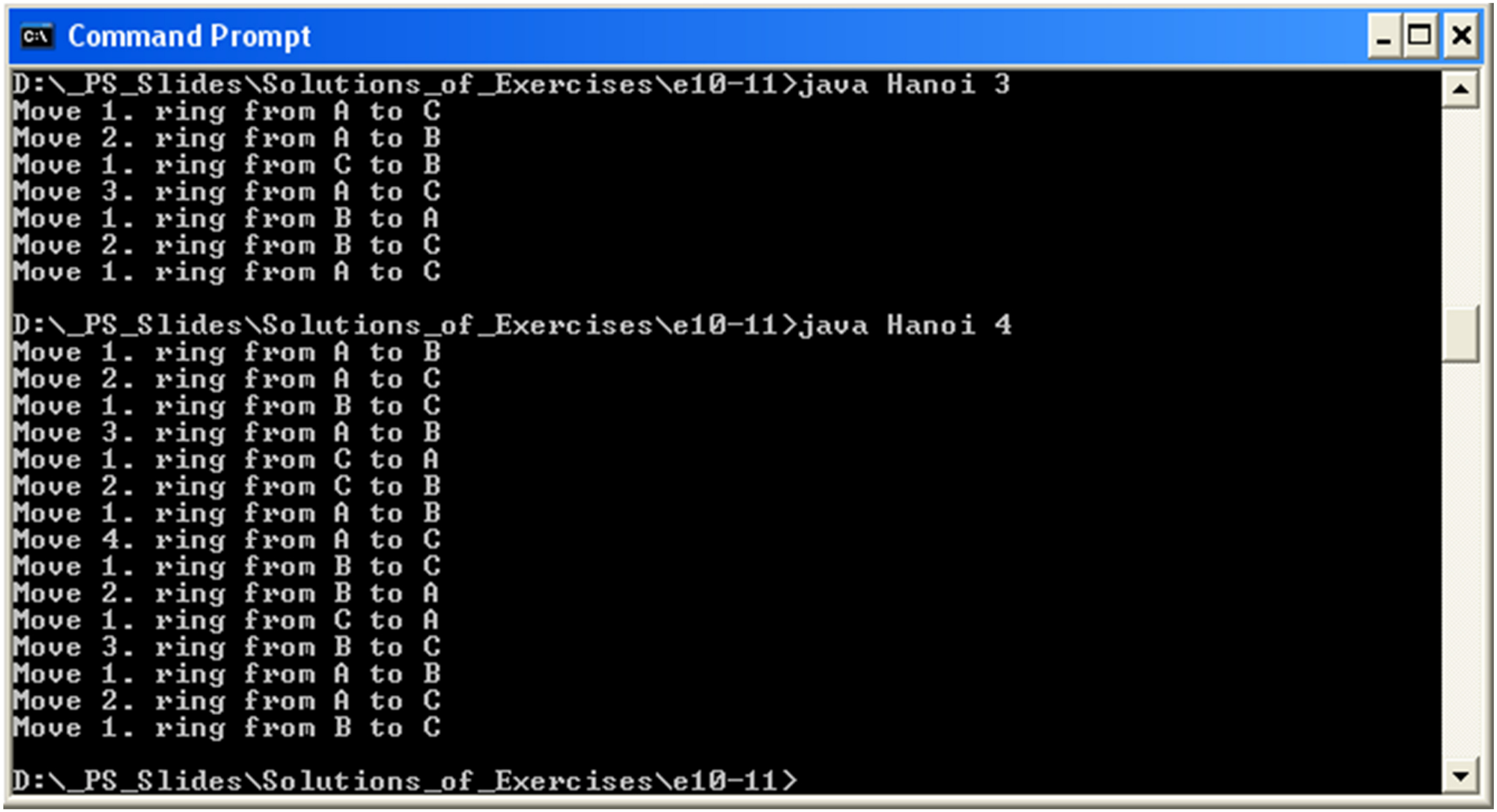


```
04/13/2008  03:58 PM    <DIR>          .
04/13/2008  03:58 PM    <DIR>          ..
04/13/2008  04:16 PM    <DIR>          stack
04/13/2008  04:24 PM    <DIR>          test
               0 File(s)              0 bytes
               4 Dir(s)   2,303,049,728 bytes free

D:\tmp\e9>java test/Test4
String Tom is inserted into the DebugStack first_stack
String Tim is inserted into the DebugStack first_stack
String Tracy is inserted into the DebugStack first_stack
String George is inserted into the DebugStack first_stack
The content of the 1. stack:Tom Tim Tracy George
String George is removed from the DebugStack first_stack
String George is inserted into the DebugStack second_stack
String Tracy is removed from the DebugStack first_stack
String Tracy is inserted into the DebugStack second_stack
String Tim is removed from the DebugStack first_stack
String Tim is inserted into the DebugStack second_stack
String Tom is removed from the DebugStack first_stack
Error: DebugStack second_stack is full
The content of the 2. stack:George Tracy Tim

D:\tmp\e9>
```

Exercise 7: Towers of Hanoi

```
class Hanoi {

        private final static char stickA = 'A';
        private final static char stickB = 'B';
        private final static char stickC = 'C';

        private static void hanoi(int n, char a, char c, char b) {
                if (n > 0) {
                        hanoi(n-1, a, b, c);
                        System.out.println("Move "+n+". ring from "+a+" to "+c);
                        hanoi(n-1, b, c, a);
                }
        }

        public static void main(String args[]) {
                if (args.length != 1) {
                        System.err.println("Usage: java Hanoi numberOfRings");
                        System.exit(1);
                }

                int i = Integer.parseInt(args[0]);
                hanoi(i, stickA, stickC, stickB);
        }
}
```

Exercise 7: Output



```
D:\_PS_Slides\Solutions_of_Exercises\e10-11>java Hanoi 3
Move 1. ring from A to C
Move 2. ring from A to B
Move 1. ring from C to B
Move 3. ring from A to C
Move 1. ring from B to A
Move 2. ring from B to C
Move 1. ring from A to C

D:\_PS_Slides\Solutions_of_Exercises\e10-11>java Hanoi 4
Move 1. ring from A to B
Move 2. ring from A to C
Move 1. ring from B to C
Move 3. ring from A to B
Move 1. ring from C to A
Move 2. ring from C to B
Move 1. ring from A to B
Move 4. ring from A to C
Move 1. ring from B to C
Move 2. ring from B to A
Move 1. ring from C to A
Move 3. ring from B to C
Move 1. ring from A to B
Move 2. ring from A to C
Move 1. ring from B to C

D:\_PS_Slides\Solutions_of_Exercises\e10-11>
```

Exercise 8: Binary Search Tree : Reminder

```
public class TreeElement {
  private int value;
  private TreeElement left, right;

  public TreeElement(int value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }

  public int getValue() { return value;}

  public TreeElement getLeft() { return left;}

  public TreeElement getRight() { return right;}

  public void setLeft(TreeElement left) {this.left=left;}

  public void setRight(TreeElement right) {this.right=right;}

  public String toString() { … }

}
```

Exercise 8: Binary Search Tree : Reminder

```
public class BinarySearchTree {

  private TreeElement root;

  public BinarySearchTree() { root = null;}

  public boolean isEmpty() { … }

  public String toString() { … }

  public void insert(int value) { … }

  public String search(int n) { …  }
}
```

Exercise 8: Binary Search Tree : search()

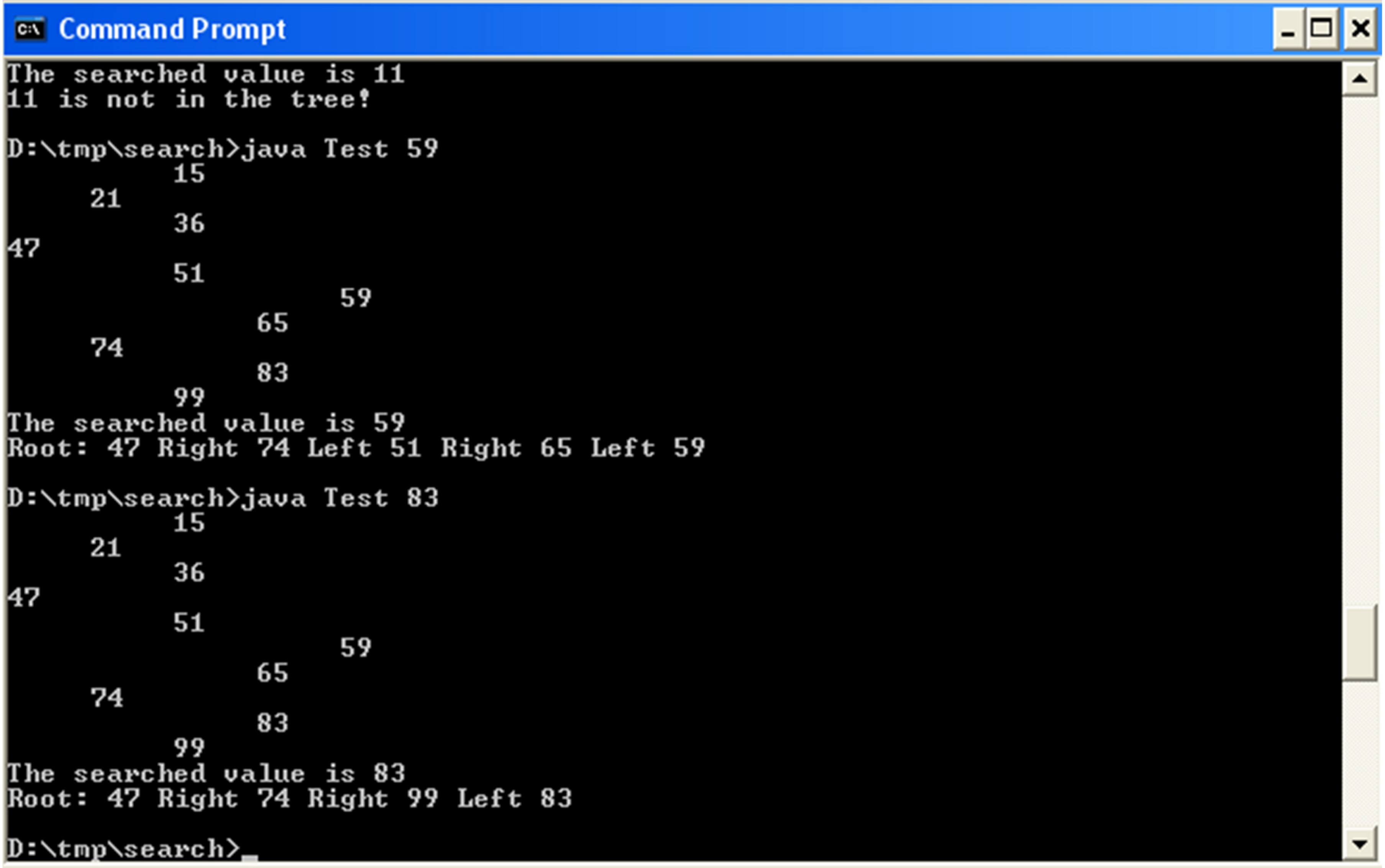
```
public String search(int n) {
    StringBuilder sb = new StringBuilder();
    sb.append("Root: ");
    TreeElement tmp = root;

    while (tmp != null && n != tmp.getValue()) {
        sb.append(tmp.getValue());
        if (n < tmp.getValue()) {
           sb.append(" Left ");
           tmp = tmp.getLeft();
        }
        else {
            sb.append(" Right ");
            tmp = tmp.getRight();
        }
    }

    if (tmp == null) { return new String(n+" is not in the tree!"); }
    else {
         sb.append(tmp.getValue());
         return sb.toString();
    }
  }
}
```

Exercise 8: Binary Search Tree : Test

```
public static void main(String[] args) {
  if (args.length !=1) {
          System.err.println("Usage: Test searchValue");
          System.exit(1);
  }
  int n = Integer.parseInt(args[0]);
  BinarySearchTree tree = new BinarySearchTree();
  tree.insert(47);
  tree.insert(74);
  tree.insert(21);
  tree.insert(99);
  tree.insert(51);
  tree.insert(15);
  tree.insert(65);
  tree.insert(36);
  tree.insert(83);
  tree.insert(59);

  //System.out.println("The content of the tree by an \"inorder\" ranging is " + tree);
  System.out.println("The searched value is "+n);
  System.out.println(tree.search(n));

 }
}
```

Exercise 8: Binary Search Tree

Karoly.Bosa@jku.at



```
The searched value is 11
11 is not in the tree!

D:\tmp\search>java Test 59
          15
     21
          36
47
          51
                    59
               65
     74
               83
          99
The searched value is 59
Root: 47 Right 74 Left 51 Right 65 Left 59

D:\tmp\search>java Test 83
          15
     21
          36
47
          51
                    59
               65
     74
               83
          99
The searched value is 83
Root: 47 Right 74 Right 99 Left 83

D:\tmp\search>_
```

Exercise 9: Reading a map from a file

```
public class Map extends Canvas {
        private final static int SQUARE_SIDE = 201;
        private final static int SQUARE_SIZE = SQUARE_SIDE * SQUARE_SIDE ;
        private int[] map = new int[SQUARE_SIZE];
        private String filename;
        private Frame frm;

…
        public void readMapFromFile() throws IOException {
                FileInputStream in = null;
                try {
                        in = new FileInputStream(filename);
                        int c;
                        int counter = 0;
                        while ((c = in.read()) != -1 && counter != SQUARE_SIZE) {
                                map[counter++] = c;
                        }

                } finally {
                        if (in != null) { in.close();}
                }
        } //ReadMapFromFile
```

Exercise 10: Generic Stack Interface

```
public interface Stack<E> {
  void clear();
  void exch() throws EmptyStackException;
  E peek() throws EmptyStackException;
  void push(E s) throws FullStackException;
  E pop() throws EmptyStackException;
  boolean isEmpty();
  String toString();
}
```

Exercise 10: Generic AbstractStack

```
public abstract class AbstractStack<E> implements Stack<E> {
    public abstract void push(E s) throws FullStackException;
    public abstract E pop() throws EmptyStackException;
    public abstract boolean isEmpty();
    public abstract String toString();

    public void clear() {
      while(!isEmpty()) { pop();}
    }

    public void exch() throws EmptyStackException {
      E tmp1 = pop();
      E tmp2 = pop();
      push(tmp1);  push(tmp2);
    }

    public E peek() throws EmptyStackException {
      E tmp = pop(); push(tmp);
      return tmp;
    }
}
```

Exercise 10: Generic BoundedStack

```
public class BoundedStack<E> extends AbstractStack<E> {
  protected E[] arrayStack;
  protected int top; //denotes the next idle place

  public BoundedStack(int size) {
    this.arrayStack = (E[])(new Object[size]);
    top = 0;
  }

  public void push(E s) throws FullStackException {
    if (top == arrayStack.length) throw new FullStackException();
    arrayStack[top++] = s;
  }

  public E pop() throws EmptyStackException {
    if (top == 0) throw new EmptyStackException();
    return arrayStack[--top];
  }
  public boolean isEmpty() { //The same as before }
  public String toString() { //The same as before }
}
```

Exercise 10: Testing Generic Stack

```
public class TestWithInteger {
  public static void main(String[] args) {
    BoundedStack<Integer> stack1 = new BoundedStack <Integer>(10);
    BoundedStack<Integer> stack2 = new BoundedStack <Integer>(10);

    stack1.push(11);
    stack1.push(22);
    stack1.push(33);
    stack1.push(44);

    System.out.println("The content of the 1. stack:" + stack1);

    while (!stack1.isEmpty()) {
      stack2.push(stack1.pop());
    }

    System.out.println("The content of the 2. stack:" + stack2);

  }
}
```

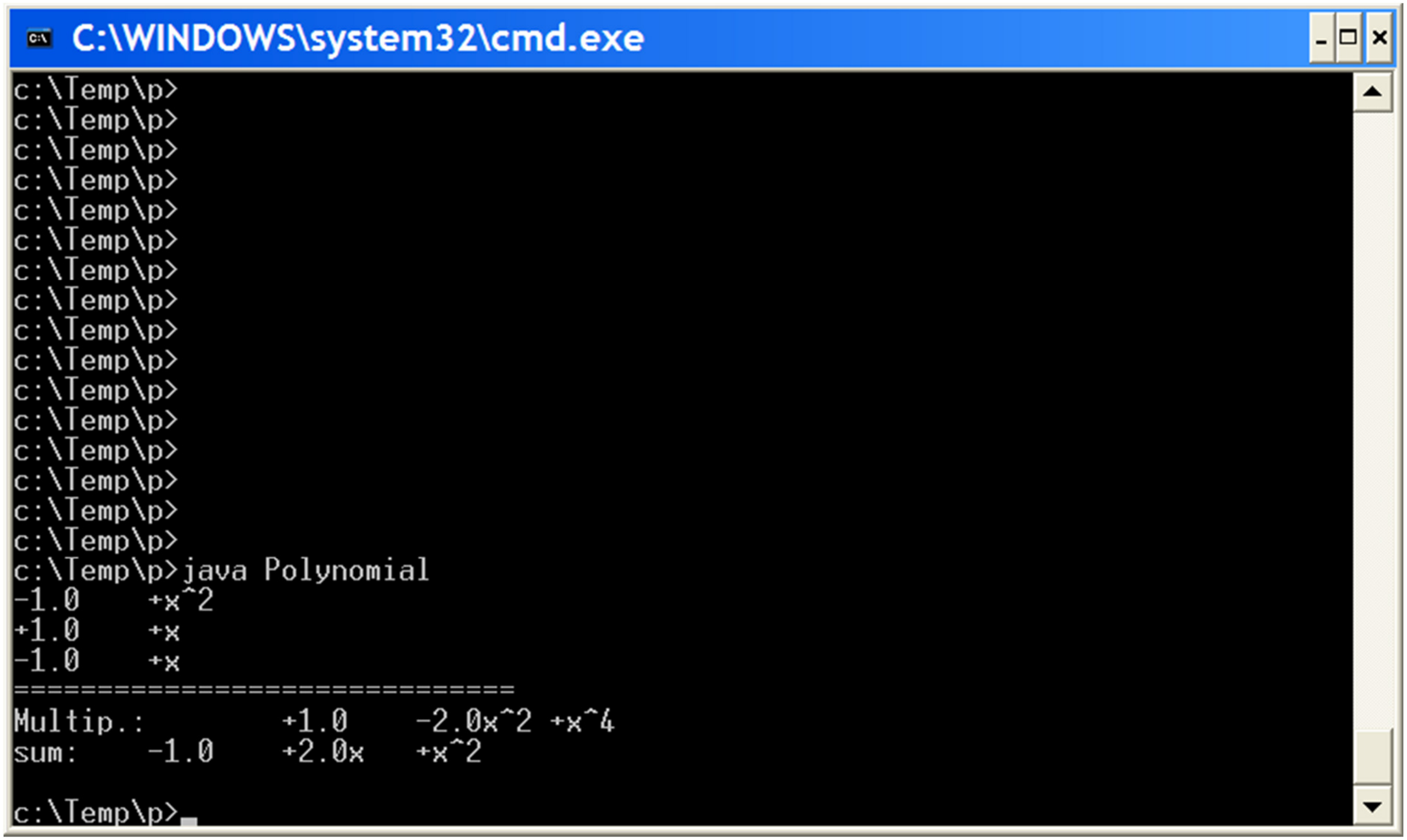
Exercise 10: Testing Generic Stack

```
public class TestWithString {
  public static void main(String[] args) {
    BoundedStack<String> stack1 = new BoundedStack<String>(10);
    BoundedStack<String> stack2 = new BoundedStack<String>(10);

    stack1.push("Tom");
    stack1.push("Tim");
    stack1.push("Tracy");
    stack1.push("George");

    System.out.println("The content of the 1. stack:" + stack1);

    while (!stack1.isEmpty()) {
      stack2.push(stack1.pop());
    }

    System.out.println("The content of the 2. stack:" + stack2);

  }
}
```

Exercise 11: Polynomial Addition

```
public static Map<Integer, Double> addition(List<Map<Integer, Double>> polynomials) {
        Map<Integer, Double> sum = new TreeMap<Integer, Double>();
        Iterator<Map<Integer, Double>>  polynomial_it = polynomials.listIterator();

        while(polynomial_it.hasNext()) {
            Map<Integer, Double> polynomial = polynomial_it.next();
            Iterator<Map.Entry<Integer, Double>> coeff_it = polynomial.entrySet().iterator();
            while (coeff_it.hasNext()) {
                Map.Entry<Integer, Double> pairs = coeff_it.next();
                double tmp_coeff = pairs.getValue().doubleValue();
                int tmp_power = pairs.getKey().intValue();
                Double current_value = sum.get(tmp_power);
                sum.put(tmp_power, (current_value == null) ? tmp_coeff : current_value + tmp_coeff);
            }
         } //while
         return sum;
}
…
List<Map<Integer, Double>> all_polynomials = new LinkedList<Map<Integer,Double>>();
Map<Integer, Double> pol1 = new TreeMap<Integer, Double>();
Map<Integer, Double> pol2 = new TreeMap<Integer, Double>();
pol1.put(2, 1.0); pol1.put(0, -1.0); all_polynomials.add(pol1);
pol2.put(1, 1.0); pol2.put(0, 1.0); all_polynomials.add(pol2);
```

Exercise 11: Output



```
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>
c:\Temp\p>java Polynomial
-1.0    +x^2
+1.0    +x
-1.0    +x
==============================
Multip.:        +1.0    -2.0x^2 +x^4
sum:    -1.0    +2.0x   +x^2

c:\Temp\p>
```