Praktische Softwaretechnologie

Lecture 6.

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

Math.round() and Math.random()

Karoly.Bosa@jku.at

Static methods of class java.lang.Math

public static long round(double a) – chooses the closer integer

public static int round(float a) – chooses the closer integer

public static double random() – generates a random floating point number between 0 and 1

Math.round() and Math.random()

Karoly.Bosa@jku.at

Usage:

- Math.round(Math.random()*100) – generates a long integer number between 0 and 100.
- Math.round(Math.random()*99+1) – generates a long integer number between 1 and 100.
- Math.round(Math.random()*100/2)*2 – generates a **EVEN** long integer number between 0 and 100.
- Math.round(Math.random()*98/2+1)*2 – generates a **EVEN** long integer number between 2 and 100.
- Math.round(Math.random()*98/2+1)*2-1 – generates an **ODD** long integer number between 0 and 100 (1-99).

Labyrinth

Karoly.Bosa@jku.at

Problem 1: Generate a labyrinth, from where there is one and only one way to get out (independently from the starting point).

Problem 2: Implement an algorithm which leads us out from the labyrinth (apply backtracking).

Labyrinth

Karoly.Bosa@jku.at

```
import java.io.*;

public class Labyrinth {

  private final static …          //Definitions of some constants

  private static void horizontal(char[][] a, int x1, int y1, int x2, int y2) {  …  }

  private static void vertical(char[][] a, int x1, int y1, int x2, int y2) {  …  }

  private static void initMatrix(char[][] a) {  ...  }

  private static void printOut(char[][] a) {   …  }

  private static boolean getOut(char[][] a, int x, int y) {   …  }

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

Labyrinth - Constants

Karoly.Bosa@jku.at

```
import java.io.*;

public class Labyrinth {
  private final static char WALL ='#';
  private final static char CORRIDOR = ' ';
  private final static char ESCAPE ='.';
  private final static int WIDTH = 75;
  private final static int HEIGHT = 25;
  private final static int EXITX = HEIGHT-2;
  private final static int EXITY = WIDTH-1;


   . . .
```

Labyrinth – method *main*


```
public static void main(String[] args) {
   char[][] a = new char[HEIGHT][WIDTH];
   initMatrix(a);
   horizontal(a, 0, 0, HEIGHT-1, WIDTH-1);

   int x = (int)(Math.round(Math.random()*((HEIGHT-1-2)/2))+1)*2-1;
   int y = (int)(Math.round(Math.random()*((WIDTH-1-2)/2))+1)*2-1;
   a[x][y]=ESCAPE;
   printOut(a);
   System.out.println("Press Enter to continue!");
   try {
     System.in.read();
   } catch (IOException e) {}
   getOut(a,x,y);
   printOut(a);
 }
```

Labirinth – method *initMatrix*

Karoly.Bosa@jku.at

```
private static void initMatrix(char[][] a) {
   for (int i=0; i<WIDTH; i++)
     a[0][i] = WALL;

   for (int j=1; j<HEIGHT-1; j++) {
     a[j][0]=WALL;
     for (int i=1; i<WIDTH-1; i++) {
       a[j][i]=CORRIDOR;
     }
     a[j][WIDTH-1]=WALL;
   }

   for (int i=0; i<WIDTH; i++)
     a[HEIGHT-1][i] = WALL;

   a[EXITX][EXITY]=CORRIDOR;
 }
```

It draws the walls of a room with one exit in the matrix.

Labyrinth – method *printOut*

Karoly.Bosa@jku.at

```
private static void printOut(char[][] a) {
   for (int j=0; j<HEIGHT; j++) {
     for (int i=0; i<WIDTH; i++) {
       System.out.print(a[j][i]);
     }
     System.out.println();
   }
 }
```

Labyrinth – method *horizontal*

Karoly.Bosa@jku.at

```
private static void horizontal (char[][] a, int x1, int y1, int x2, int y2) {
    int j = (int)(Math.round(Math.random()*((y2-y1-4)/2))+1)*2;
    for(int i=x1; i<=x2; i++) {
      a[i][y1+j]=WALL;
    }
    int i = (int)(Math.round(Math.random()*((x2-x1-2)/2))+1)*2-1;
    a[x1+i][y1+j]=CORRIDOR;

    if (j > 2) vertical(a, x1, y1, x2, y1+j);
    if (y2-y1-j > 2) vertical(a, x1, y1+j, x2, y2);
  }
```

It divides a (sub)room to two parts with a vertical line/wall and open a gateway on it.

Labyrinth – method *vertical*

Karoly.Bosa@jku.at

```
private static void vertical(char[][] a, int x1, int y1, int x2, int y2) {
    int j = (int)(Math.round(Math.random()*((x2-x1-4)/2))+1)*2;
    for(int i=y1; i<=y2; i++) {
      a[x1+j][i]=WALL;
    }
    int i = (int)(Math.round(Math.random()*((y2-y1-2)/2))+1)*2-1;
    a[x1+j][y1+i]=CORRIDOR;

    if (j > 2) horizontal(a, x1, y1, x1+j, y2);
    if (x2-x1-j > 2) horizontal(a, x1+j, y1, x2, y2);
  }
```

It divides a (sub)room to two parts with a horizontal line/wall and open a gateway on it.

Labyrinth – method *getOut*

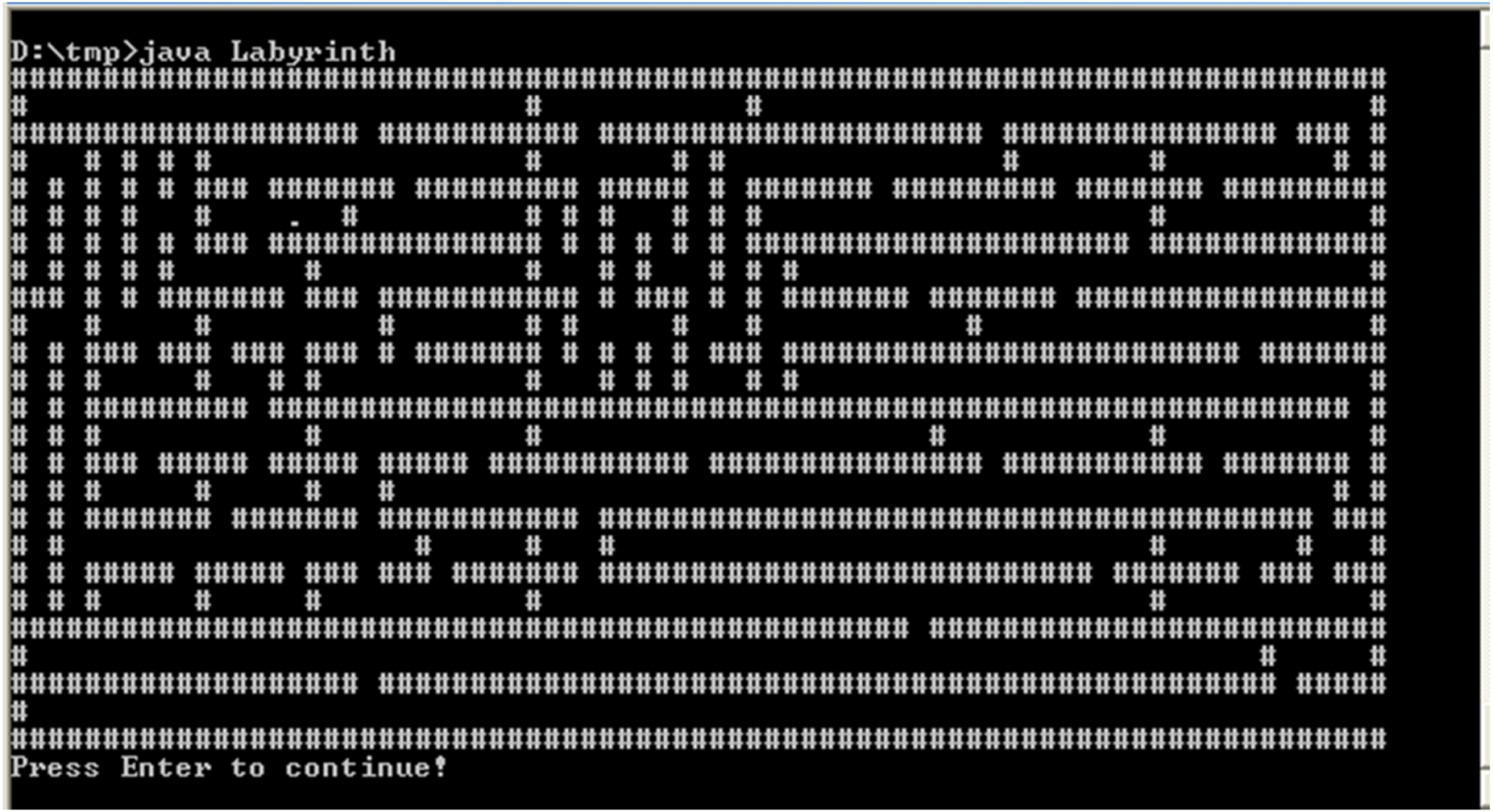
Karoly.Bosa@jku.at

```
private static boolean getOut(char[][] a, int x, int y) {
    if (x==0 || y==0 || x==HEIGHT-1 || y==WIDTH-1) {
      return true;
    }

    boolean success = false;
    a[x][y] = ESCAPE;

    if (a[x-1][y] == CORRIDOR) success = getOut(a, x-1, y);
    if (!success && a[x][y-1] == CORRIDOR) success = getOut(a, x, y-1);
    if (!success && a[x+1][y] == CORRIDOR) success = getOut(a , x+1, y);
    if (!success && a[x][y+1] == CORRIDOR) success = getOut(a, x, y+1);

    if (!success) a[x][y]=CORRIDOR; // tidy up

    return success;
  }
```

Typical application of backtracking!

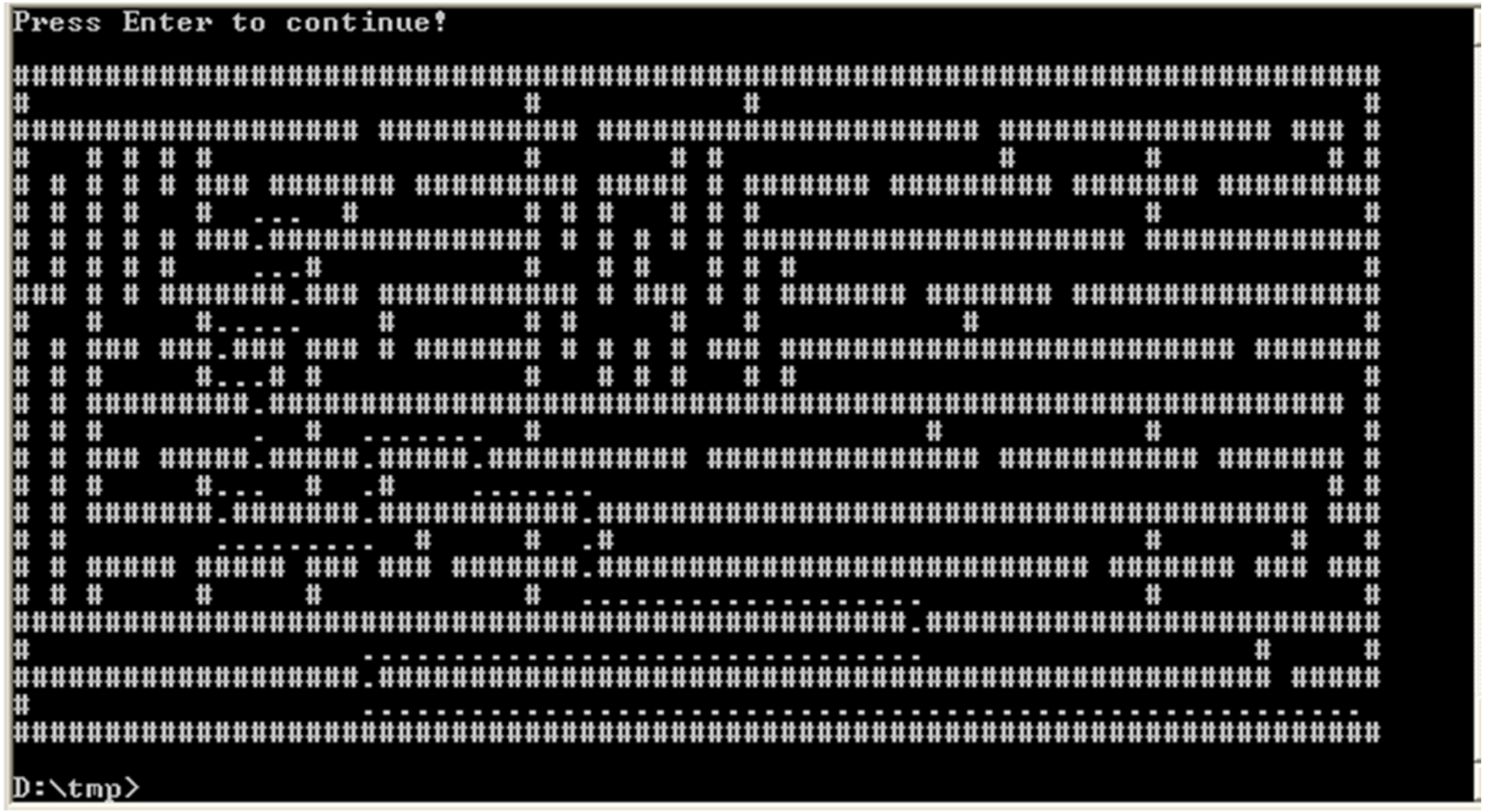
Labyrinth – Output 1.

Karoly.Bosa@jku.at



Labyrinth – Output 2.

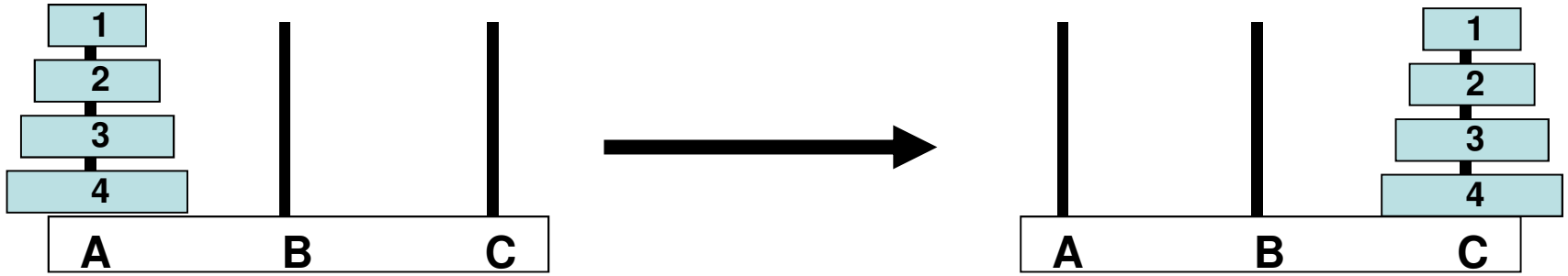
Karoly.Bosa@jku.at



Exercise 7 – Towers of Hanoi

Deadline: 30.04.2014

Karoly.Bosa@jku.at



You must place all rings from the stick A to the stick C in the same order with the help of stick B, such that:

- You can move only one ring in one step from a stick to another
- You can put a ring only either on an empty stick or on a top of another ring whose diameter is bigger.
- The program prints out the movement of a ring in each step, e.g.: "Move the 3. ring from A to B!"

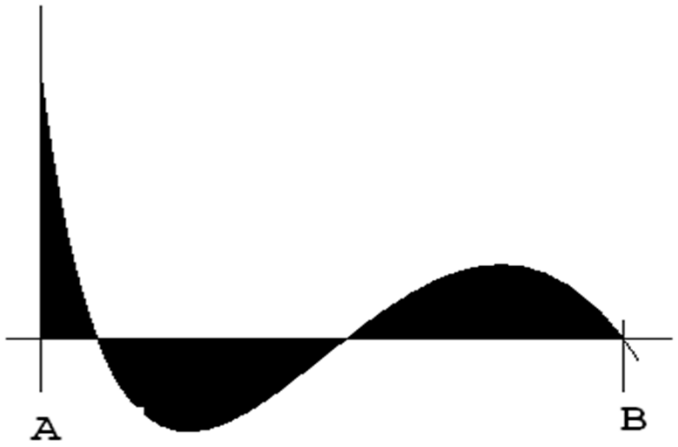
You must implement a recursive algorithms for the problem.

Definite Integral

Karoly.Bosa@jku.at

Problem: Compute (more or less precisely) the definite integral of function sin(x)/x between 0.1 and 10.

How: The value of the definite integral of a function within a predefined interval is equal to the area under of the function curve ➔ We will approach this area with areas of trapezes.

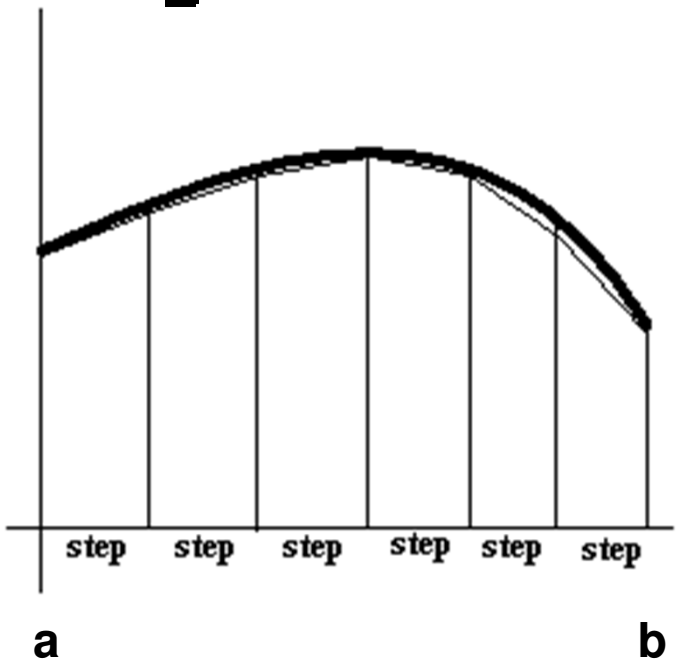


Definite Integral

Karoly.Bosa@jku.at

$$T_n = \frac{f(a)+f(a+s)}{2}s + \frac{f(a+s)+f(a+2s)}{2}s + \ldots + \frac{f(a+(n-1)s)+f(b)}{2}s$$

$$T_n = s\left[\frac{f(a)}{2} + f(a+s) + f(a+2s) + \ldots + f(a+(n-1)s) + \frac{f(b)}{2}\right]$$



Definite Integral

Karoly.Bosa@jku.at

```
public class DefiniteIntegral1 {
  public static void main(String args[]) {
    double bottomX = 0.1;
    double topX= 10;
    double area = (Math.sin(bottomX)/bottomX + Math.sin(topX)/topX) / 2;

    if (args.length == 0) return;
    int n = Integer.parseInt(args[0]);

    double step = (topX-bottomX)/n;
    double x = bottomX + step;

    for (int k = 1; k < n; k++) {
       area += Math.sin(x)/x;
       x += step;
    }
    area *= step;
    System.out.println("The definite integral of sin(x)/x between " + bottomX + "
and " + topX + " is " + area);
  }
}
```

Definite Integral - Output

Karoly.Bosa@jku.at

```
D:\>cd tmp

D:\tmp>java DefiniteIntegral1 10
The definite integral of sin(x)/x between 0.1 and 10.0 is 1.4107173602374352

D:\tmp>java DefiniteIntegral1 100
The definite integral of sin(x)/x between 0.1 and 10.0 is 1.5468605464223368

D:\tmp>java DefiniteIntegral1 1000
The definite integral of sin(x)/x between 0.1 and 10.0 is 1.5573179459717392

D:\tmp>java DefiniteIntegral1 10000
The definite integral of sin(x)/x between 0.1 and 10.0 is 1.5582953363702468

D:\tmp>java DefiniteIntegral1 100000
The definite integral of sin(x)/x between 0.1 and 10.0 is 1.5583923606852785

D:\tmp>java DefiniteIntegral1 1000000
The definite integral of sin(x)/x between 0.1 and 10.0 is 1.5584020559400473

D:\tmp>java DefiniteIntegral1 10000000
The definite integral of sin(x)/x between 0.1 and 10.0 is 1.558403025326189

D:\tmp>_
```

Definite Integral 2 with Given Precision

Karoly.Bosa@jku.at

Problem: Compute the definite integral of function sin(x)/x between 0.1 and 10 with a given decimal precision.

Definite Integral 2

Karoly.Bosa@jku.at

```
public class DefiniteIntegral2 {
  public static void main(String args[]) {
    double bottomX = 0.1;
    double topX= 10;
    double area = 0;
    double prevArea;
    int n = 10;

    if (args.length == 0) return;
    double precision = Double.parseDouble(args[0]);
    …



}
```

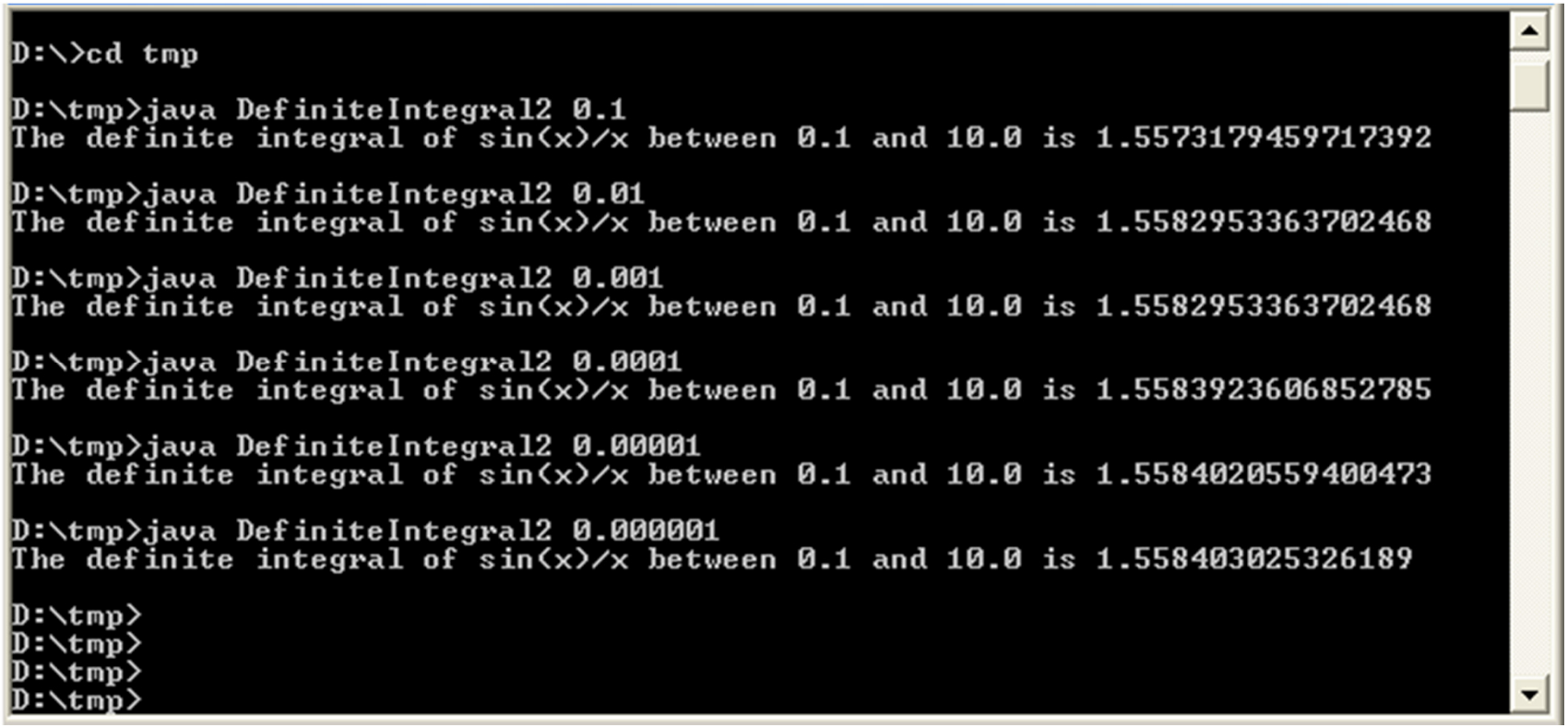
Definite Integral 2

Karoly.Bosa@jku.at

```
do {
    prevArea = area;

    area = (Math.sin(bottomX)/bottomX + Math.sin(topX)/topX) / 2;
    double step = (topX-bottomX)/n;
    double x = bottomX + step;

    for (int k = 1; k < n; k++) {
      area += Math.sin(x)/x;
      x += step;
    }
    area *= step;
    n *= 10;
  } while (Math.abs(area-prevArea) > precision);

  System.out.println("The definite integral of sin(x)/x between " + bottomX + "
and " + topX + " is " + area);
 }
```

Definite Integral 2 - Output

Karoly.Bosa@jku.at



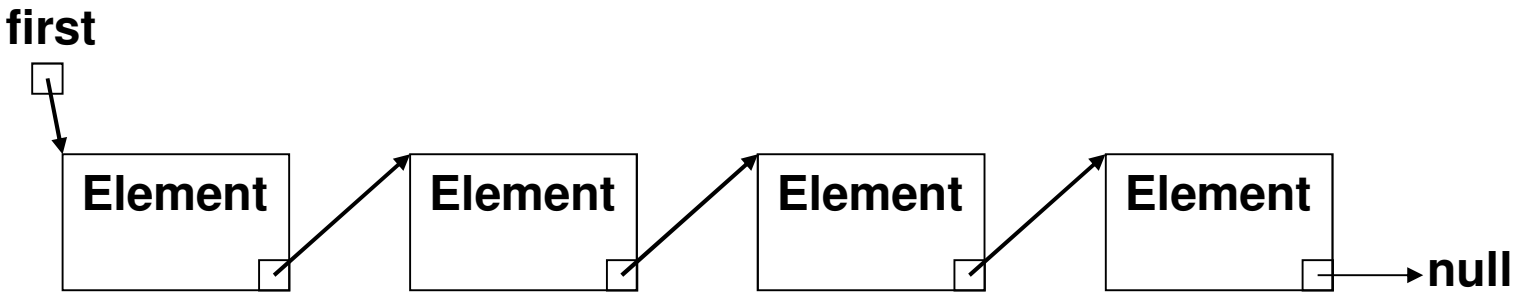
Stack Made from a Linked List

Karoly.Bosa@jku.at

Problem: Create a stack where the maximum number of its element is not predefined.

How: Apply a list of objects, where an object always refers to the subsequent object via one of its data field.

Another advantages: Memory is allocated only for the existing elements.



Stack Made from a Linked List

Karoly.Bosa@jku.at

```
public class StackElement {
  private String value;
  private StackElement next;

  public StackElement(String value, StackElement next) {
    this.value = value;
    this.next = next;
  }

  public String getValue() {
    return value;
  }

  public StackElement getNext() {
    return next;
  }
}
```

Stack Made from a Linked List

Karoly.Bosa@jku.at

```
public class LinkedListStack {

  private StackElement top;

  public LinkedListStack() {
    top = null;
  }

  public void push(String s) {
    StackElement tmp = new StackElement(s, top);
    top = tmp;
  }

  public String pop() {
    StackElement tmp = top;
    top = tmp.getNext();
    return tmp.getValue();
  }

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

  public String toString() {
    StringBuilder sb =new StringBuilder();
    StackElement tmp = top;
    while (tmp !=null) {
      sb.append(tmp.getValue()+" ");
      tmp = tmp.getNext();
    }
    return sb.toString();
  }

}
```

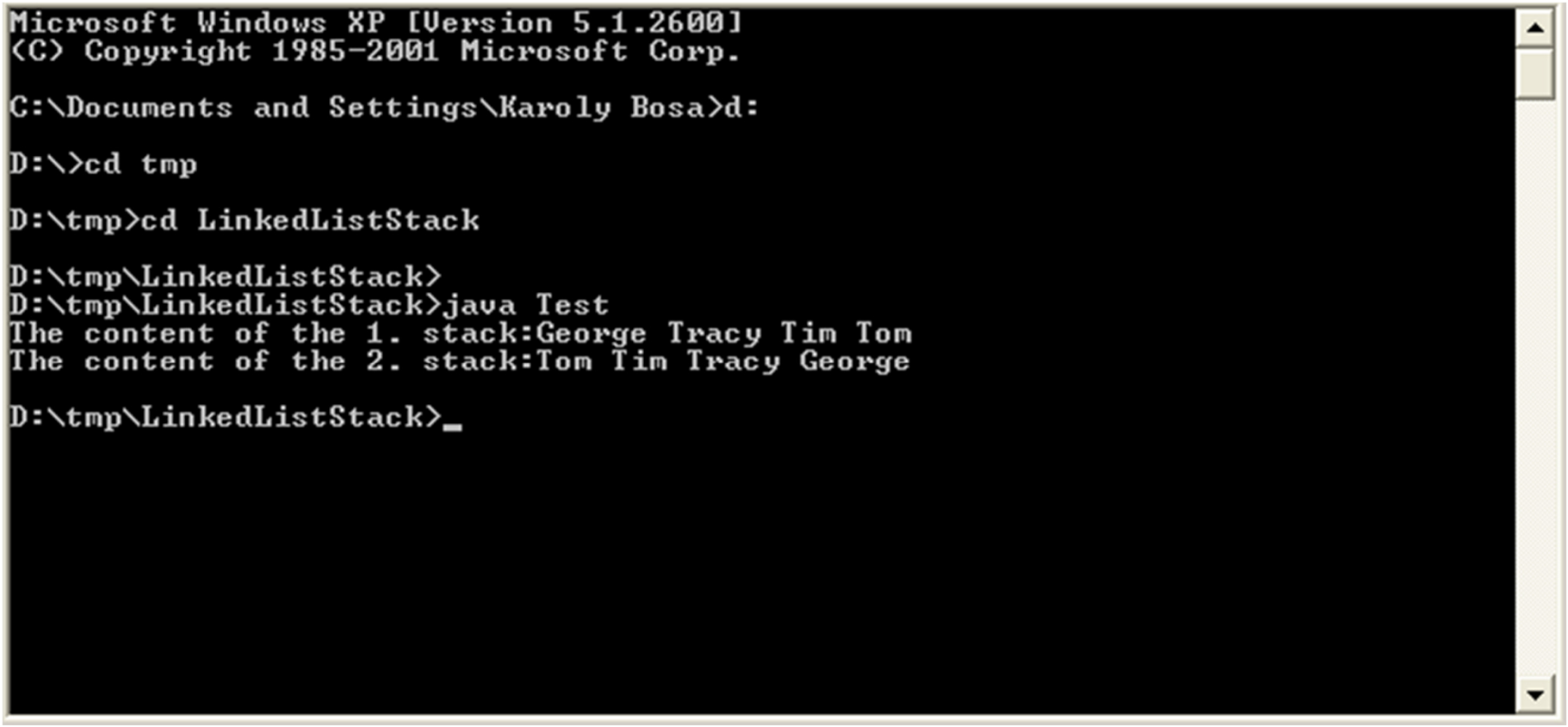
Stack Made from a Linked List

Karoly.Bosa@jku.at

```
public class Test {
  public static void main(String[] args) {
    LinkedListStack stack1 = new LinkedListStack();  //no given size
    LinkedListStack stack2 = new LinkedListStack();  //no given size

    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);

  }
}
```

Stack Made from a Linked List - Output

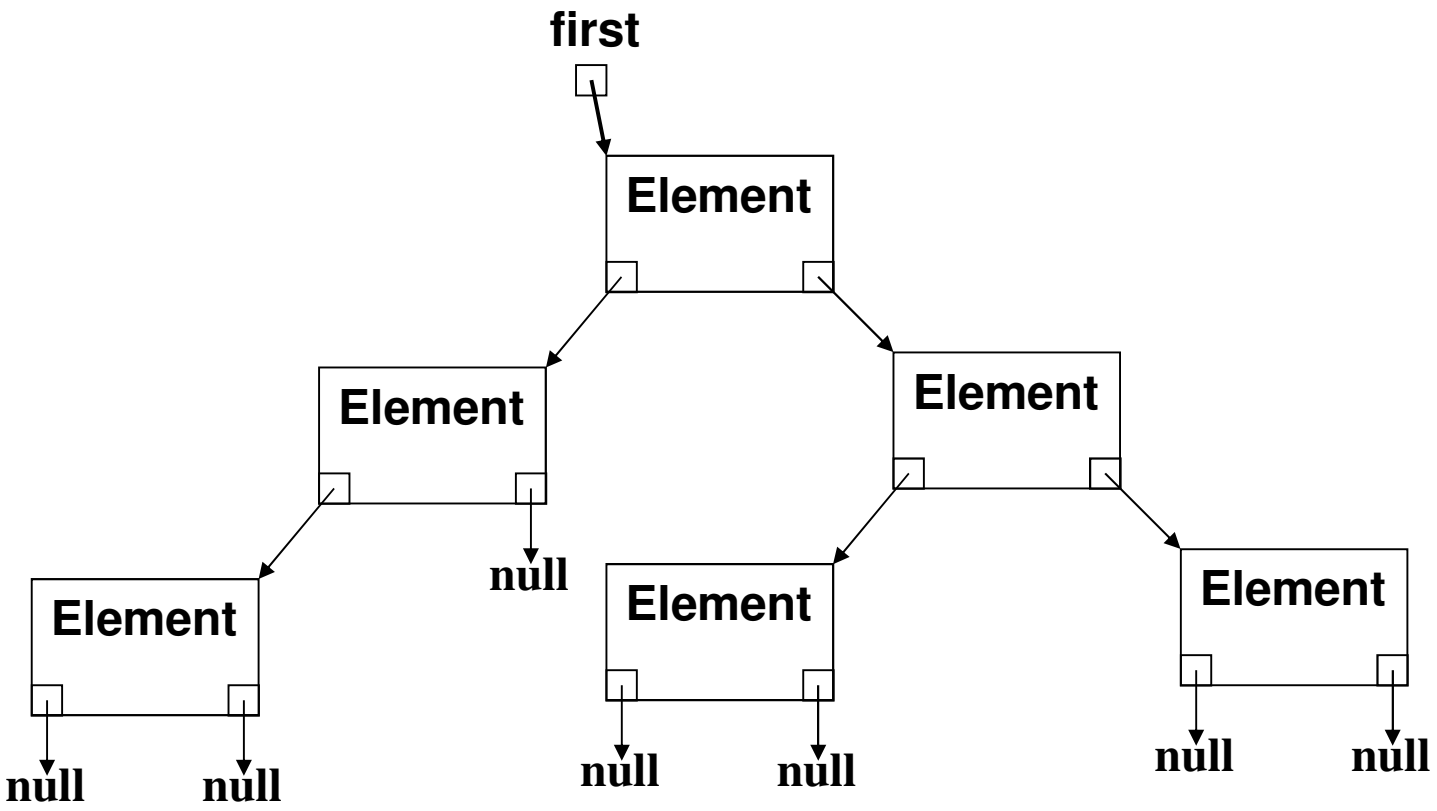
Karoly.Bosa@jku.at



Binary Trees

Karoly.Bosa@jku.at

A linked data structure is called binary tree, if each element is followed at most two subsequent elements.

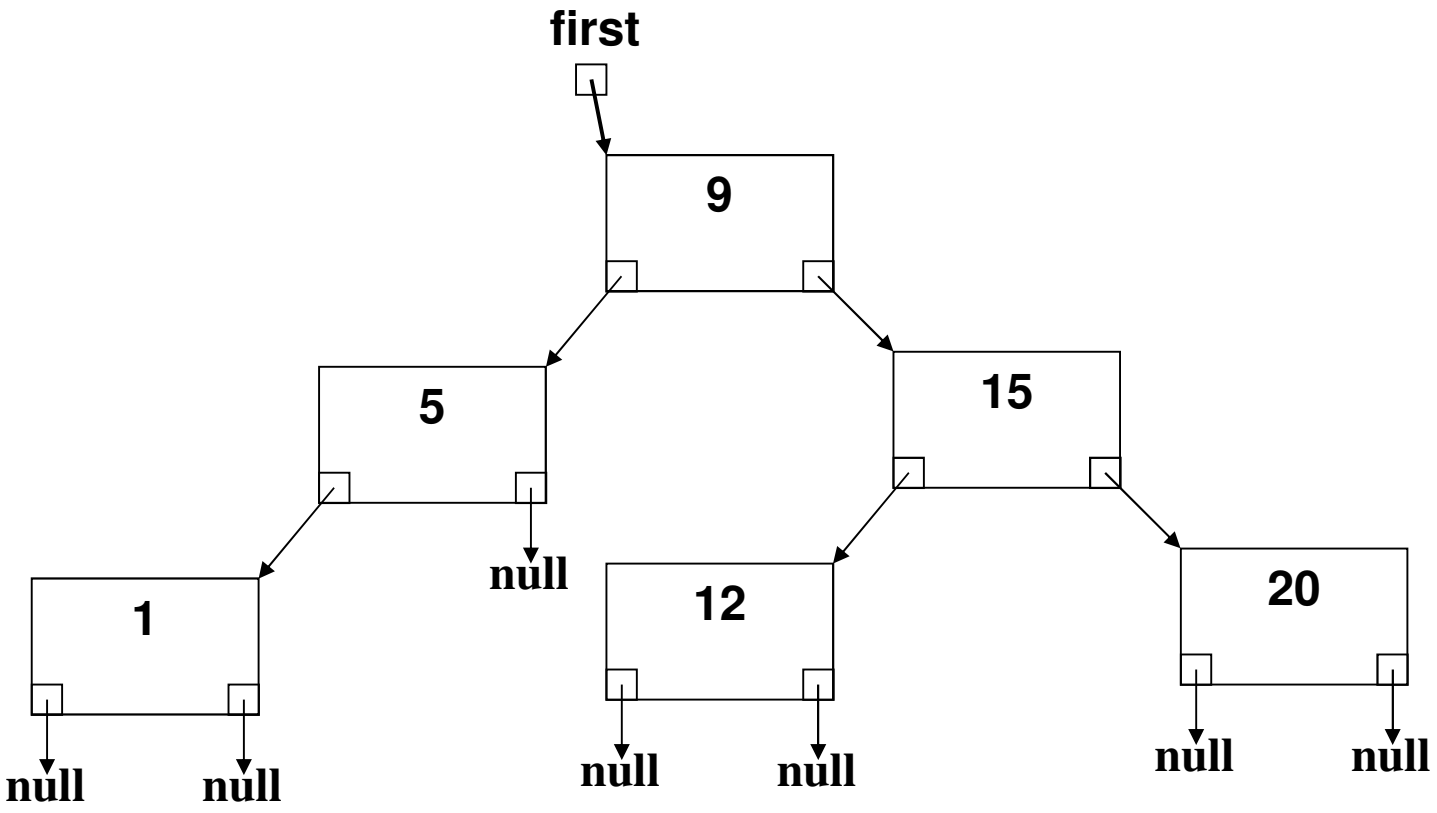


Binary Search Trees

Karoly.Bosa@jku.at

A binary tree is a search tree, if each node fulfills that:

- its value is greater than all values stored in its left sub-tree.
- its value is less than all values stored in its right sub-tree.



Binary Search Trees – class TreeElement

Karoly.Bosa@jku.at

```
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() {
    StringBuilder sb = new
                        StringBuilder();
    if (left != null)
          sb.append(left.toString());
    sb.append(value+" ");
    if (right != null)
          sb.append(right.toString());
    return sb.toString();
  }
}
```

Binary Search Trees – class BinarySearchTree

Karoly.Bosa@jku.at

```
public class BinarySearchTree {

 private TreeElement root;

 public BinarySearchTree() {
  root = null;
 }

 public boolean isEmpty() {
  if (root==null) return true;
  return false;
 }

 public String toString() {
    if (root != null) {
    return root.toString();
  }
  return "";
 }
```

Binary Search Trees – class BinarySearchTree

Karoly.Bosa@jku.at

```
public void insert(int value) {
   boolean inserted = false;
   TreeElement newElement = new TreeElement(value);
   if (root==null) { root = newElement;}
   else {
     TreeElement tmp = root;
     while (!inserted) {

       if (tmp.getValue() == value) {inserted = true;}
       else if (tmp.getValue() > value) {
              if (tmp.getLeft() != null) {tmp = tmp.getLeft();}
              else { tmp.setLeft(newElement); inserted = true;}
       }
       else {
              if (tmp.getRight() != null) {tmp = tmp.getRight();}
              else { tmp.setRight(newElement); inserted = true;}
       }
     }
   }
 }
}
```

Binary Search Trees – class Test

Karoly.Bosa@jku.at

```
public class Test {
  public static void main(String[] args) {
    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);

  }
}
```

Exercise 8 – Extend the Search Tree

Karoly.Bosa@jku.at

Extend the class *BinarySearchTree* with a method *String search(int n),* which search n in the tree. The method returns the description, how to get to n.

E.g.:

Input: n = 11

Output: "Root:16 Left 5 Right 10 Right 11"

Test the *search() method*!

Deadline: 30.04.2014