Praktische Softwaretechnologie

Lecture 5.

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

The “Wrapper” Classes

There is one class in the java.lang for each primitive type

Boolen, Byte, Short, Integer, Long, Character, Float, Double

Each object contains an non-modifiable field of the corresponding type

The most important fields/methods:

- MIN_VALUE, MAX_VALUE
- int intValue(), etc.
- static Integer valueOf(int)
- static int parseInt(String)
- In Character: toLower(), toUpper(), etc.

Autoboxing

Karoly.Bosa@jku.at

Conversion between primitive Wrapper type on demand:

```
void mPrim(int i) {...}
void mRef(Integer i) {...}

int prim = 42;
Integer ref = new Integer(23);
```

Compiler changes:

`mPrim(ref)` **as well as** `mRef(42)`

to

`mPrim(ref.intValue())` **as well as** `mRef(Integer.valueOf(42))`

More about Strings

Karoly.Bosa@jku.at

Most important methods:

- `char charAt(int index), String substring(...)`
- `boolean startsWith(String start), boolean endsWith(String end)`
- `int indexOf(String str), ...`
- `String toLowerCase(), String toUpperCase()`

StringBuilder


```
String s = "";
for (int i=0; i<100; i++) {
   s = s + text(i);
}
```

StringBuilder


```
String s = "";
for (int i=0; i<100; i++) {
   s = s + text(i);  // 100 neue String Objekte!
}
```

StringBuilder


```
String s = "";
for (int i=0; i<100; i++) {
   s = s + text(i);  // 100 neue String Objekte!
}
```

Better: Build an Object with StringBuilder and then declare a String

```
StringBuilder sb = new StringBuilder();
for (int i=0; i<100; i++) {
   sb.append(text(i));
}

String s = sb.toString();
```

StringBuilder


```
String s = "";
for (int i=0; i<100; i++) {
   s = s + text(i);  // 100 neue String Objekte!
}
```

Better: Build an Object with StringBuilder and then declare a String

```
StringBuilder sb = new StringBuilder();
for (int i=0; i<100; i++) {
   sb.append(text(i));
}

String s = sb.toString();
```

Concatenation :

```
sb.append("Text Nr.").append(i).append(" ist ").append(text(i));
```

Converting String to a primitive type

Casting will not work:

```
int n = (int)”123”;
```

But:

```
byte b = Byte.parseByte("123");
short s = Short.parseShort("123");
int i = Integer.parseInt("123");
long l = Long.parseLong("123");
float f = Float.parseFloat("123.4");
double d = Double.parseDouble("123.4e10");
```

Arguments of the Methods

This will not work:

```
public class Argument {

 private static void increment(int n) {
   n++;
 }

 public static void main (String[] args) {

    int n = 0;
    increment(n);
    System.out.println("The value of n is still: "+n);
 }
}
```

In case of arrays and objects we give their references to the methods. These references point to somewhere in the memory. **Therefore, the values of arrays and object can be modified from methods!!!**

Linear Search

Problem: Searching for an element in an unsorted list

In the worst case: It finds the searched element only after n step in a list which consists of n elements.

Linear Search


```
public class LinealSearch {

 private static int search(int[] a, int n) {
   for (int i=0; i<a.length; i++) {
     if (a[i]==n) return i;
   }
   return -1;
 }

 public static void main (String[] args) {
    int[] arrayOfNumbers = {47, 74, 21, 99, 51, 15, 65, 36, 83, 59};

    if (args.length == 0) return;
    int n = Integer.parseInt(args[0]);
    int pos = search(arrayOfNumbers, n);
    if (pos == -1)
      System.out.println("Sorry! But "+ n +" does not occur in the array!");
    else
      System.out.println("The position of "+n+" in the array is: " + pos);
 }
}
```

Binary Search

Karoly.Bosa@jku.at

Problem: Searching for an element in an sorted list

Requirement: We can apply it only on a list, which was sorted in advance.

In the worst case: It finds the searched element only after $\log_2 n+1$ step in a sorted list which contsists of n elements.

Binary Search

Karoly.Bosa@jku.at

```
public class BinarySearch {

 private static int search(int[] a, int n) {
            …
 }

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

}
```

Binary Search – method *main*


```
public static void main (String[] args) {
    int[] arrayOfNumbers = {15, 21, 36, 47, 51, 59, 65, 74, 83, 99};

    if (args.length == 0) return;
    int n = Integer.parseInt(args[0]);
    int pos = search(arrayOfNumbers, n);
    if (pos == -1)
      System.out.println("Sorry! But "+ n +" does not occur in the array!");
    else
      System.out.println("The position of "+n+" in the array is: " + pos);
 }
```

Binary Search – method *search*


```
private static int search(int[] a, int n) {
   int lowerIndex=0;
   int upperIndex=a.length-1;
   int middleIndex= (lowerIndex+upperIndex)/2;

   while (lowerIndex<=upperIndex && a[middleIndex]!=n) {

     if (a[middleIndex]>n) upperIndex=middleIndex-1;
     else if (a[middleIndex]<n) lowerIndex=middleIndex+1;

     middleIndex= (lowerIndex+upperIndex)/2;
   }
   if (lowerIndex<=upperIndex) return middleIndex;

   return -1;
 }
```

Bubble Sort

Problem: Sorting an array. There exist many algorithms for this problem, e.g. bubble sort, inserting sort, shell sort, quick sort, etc.

One of the simplest one is the bubble sort

Bubble Sort


```
public class BubbleSort {

 public static void main(String[] args) {
  int[] arrayOfNumbers = {47, 74, 21, 99, 51, 15, 65, 36, 83, 59};

  int tmp;
  boolean sorted = false;

  for (int i = arrayOfNumbers.length-1; i >= 1 && !sorted ; i--) {
   sorted = true;

   for (int j = 0; j <= i-1; j++) {
    if (arrayOfNumbers[j] > arrayOfNumbers[j+1]) {
            tmp = arrayOfNumbers[j];
            arrayOfNumbers[j] = arrayOfNumbers[j+1];
            arrayOfNumbers[j+1] = tmp;
            sorted = false;
          }
   }
  }
```

Bubble Sort (continuation)


```
for (int i = 0; i < arrayOfNumbers.length; i++) {
     System.out. print(arrayOfNumbers[i]+" ");
   }
   System.out.println();
  }
}
```

Bubble sort - Output


```
kbosa@vm5:~/tmp$ java BubbleSort
0. step: 47 74 21 99 51 15 65 36 83 59
1. step: 47 21 74 99 51 15 65 36 83 59
2. step: 47 21 74 51 99 15 65 36 83 59
3. step: 47 21 74 51 15 99 65 36 83 59
4. step: 47 21 74 51 15 65 99 36 83 59
5. step: 47 21 74 51 15 65 36 99 83 59
6. step: 47 21 74 51 15 65 36 83 99 59
7. step: 47 21 74 51 15 65 36 83 59 99
8. step: 21 47 74 51 15 65 36 83 59 99
9. step: 21 47 51 74 15 65 36 83 59 99
10. step: 21 47 51 15 74 65 36 83 59 99
11. step: 21 47 51 15 65 74 36 83 59 99
12. step: 21 47 51 15 65 36 74 83 59 99
13. step: 21 47 51 15 65 36 74 59 83 99
14. step: 21 47 15 51 65 36 74 59 83 99
15. step: 21 47 15 51 36 65 74 59 83 99
16. step: 21 47 15 51 36 65 59 74 83 99
17. step: 21 15 47 51 36 65 59 74 83 99
18. step: 21 15 47 36 51 65 59 74 83 99
19. step: 21 15 47 36 51 59 65 74 83 99
20. step: 15 21 47 36 51 59 65 74 83 99
21. step: 15 21 36 47 51 59 65 74 83 99
kbosa@vm5:~/tmp$
```

Recursion

Karoly.Bosa@jku.at

Recursion ➔ something contains itself

Direct recursion: A method calls itself
Indirect recursion: Two or more methods call each other

Termination condition: We must always define a termination condition precisely (if this holds the method does not call itself anymore), otherwise we can easily run out from the memory (Stack Overflow).

If a problem can be solved recursively, then it has a non-recursive solution, too (but the recursive one is usually shorter and easier – similar to *induction*).

Computing Factorials

Karoly.Bosa@jku.at

```
public class RecursiveFact {
    private static int fact(int n) {
       …
    }

    public static void main (String[] args) {
       if (args.length == 0) return;
       int n = Integer.parseInt(args[0]);
       System.out.println(n + ". factorial is " + fact(n));
    }
}
```

Computing Factorials

Karoly.Bosa@jku.at

Conventional solution:

```
private static int fact(int n) {
        int f=1;
        for (int i=1; i<=n; i++) {
            f*=i;
        }
        return f;
    }
```

Recursive solution:

```
private static int fact(int n) {
        if (n==0) return 1;
        else return n*fact(n-1);
    }
```

Fibonacci Sequence

Fibonacci sequence:

$$Fib_0 = 0$$
$$Fib_1 = 1$$
$$Fib_n = Fib_{n-1} + Fib_{n-2}\text{ , where } n>1$$

E.g.: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,..

Problem: We give the position of an element in the Fibonacci sequence and the program must returns with the corresponding element.

Fibonacci Sequence


```
public class RecursiveFibonacci {
  private static int fib(int n) {
    switch(n) {
        case 0: return 0;
        case 1: return 1;
        default: return fib(n-1)+fib(n-2);
    }
  }

  public static void main (String[] args) {
   if (args.length == 0) return;
   int n = Integer.parseInt(args[0]);
   System.out.println(n + ". fibonacci number is " + fib(n-1));
  }
}
```

Fibonacci Sequence - Output

Karoly.Bosa@jku.at

```
kbosa@vm5:~$ cd course/
kbosa@vm5:~/course$ java RecursiveFibonacci 1
1. fibonacci number is 0
kbosa@vm5:~/course$ java RecursiveFibonacci 2
2. fibonacci number is 1
kbosa@vm5:~/course$ java RecursiveFibonacci 5
5. fibonacci number is 3
kbosa@vm5:~/course$ java RecursiveFibonacci 10
10. fibonacci number is 34
kbosa@vm5:~/course$ java RecursiveFibonacci 15
15. fibonacci number is 377
kbosa@vm5:~/course$ java RecursiveFibonacci 16
16. fibonacci number is 610
kbosa@vm5:~/course$ java RecursiveFibonacci 20
20. fibonacci number is 4181
kbosa@vm5:~/course$
```

Quick Sort

Problem: Sorting an array. Quick sort is the most efficient sorting algorithm.

Quick Sort


```
public class QuickSort {

  private static void quick(int[] a, int lower, int upper) {
    …
  }

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

Quick Sort – method *main*


```
public static void main(String[] args) {
   int[] arrayOfNumbers = {47, 74, 21, 99, 51, 15, 65, 36, 83, 59};

   quick(arrayOfNumbers, 0, arrayOfNumbers.length-1);

   for (int i = 0; i < arrayOfNumbers.length; i++) {
     System.out. print(arrayOfNumbers[i]+" ");
   }
   System.out.println();
 }
```

Quick Sort – method *quick*


```
private static void quick(int[] a, int lower, int upper) {
   int middleValue = a[(lower+upper)/2];
   int i = lower;
   int j = upper;
   int tmp;

   do {
     while (a[i] < middleValue) { i++; }
     while (a[j] > middleValue) { j--; }

     if (i < j) {
        tmp = a[i]; a[i] = a[j]; a[j] = tmp;
     }

     if (i<=j) { i++; j--;}
   } while (i<=j);

   if (lower<j) quick(a, lower, j);
   if (i<upper) quick(a, i, upper);
}
```

Quick Sort - Output


```
kbosa@vm5:~$ cd tmp
kbosa@vm5:~/tmp$ java QuickSort
0. step: 47 74 21 99 51 15 65 36 83 59
1. step: MiddleValue: 51
     47 36 21 99 51 15 65 74 83 59

2. step: MiddleValue: 51
     47 36 21 15 51 99 65 74 83 59

3. step: MiddleValue: 36
     15 36 21 47 51 99 65 74 83 59

4. step: MiddleValue: 36
     15 21 36 47 51 99 65 74 83 59

5. step: MiddleValue: 74
     15 21 36 47 51 59 65 74 83 99

kbosa@vm5:~/tmp$
```

Backtracking

Problem: We have a set of variables. We must assign to each of them a value from a finite interval, such that some conditions have to be fulfilled. Backtracking attempts to try all the combinations of the values in order to obtain a solution.

Backtracking algorithms try each possibility until they find the right one. It is a depth-first search algorithm.

During the search, if an alternative doesn't work, the search backtracks to the *last choice point*, the place which presented different alternatives, and tries the next alternative.

When the alternatives are exhausted, the search returns to the previous choice point and tries the next alternative there. If there are no more choice points, the search fails.

8 Queens

Problem: How to place 8 queens on chess-board, such that they do not capture each other.

Solution: with backtracking

8 Queens (How can two queens hit each other)

Karoly.Bosa@jku.at

1.) They are in the same row.

Q_j	.	.	.	.	Q_k		

It cannot occur in our vector representation

(each element of the array contains the position of a queen in the corresponding row, therefore we cannot place more than one queen into one row).

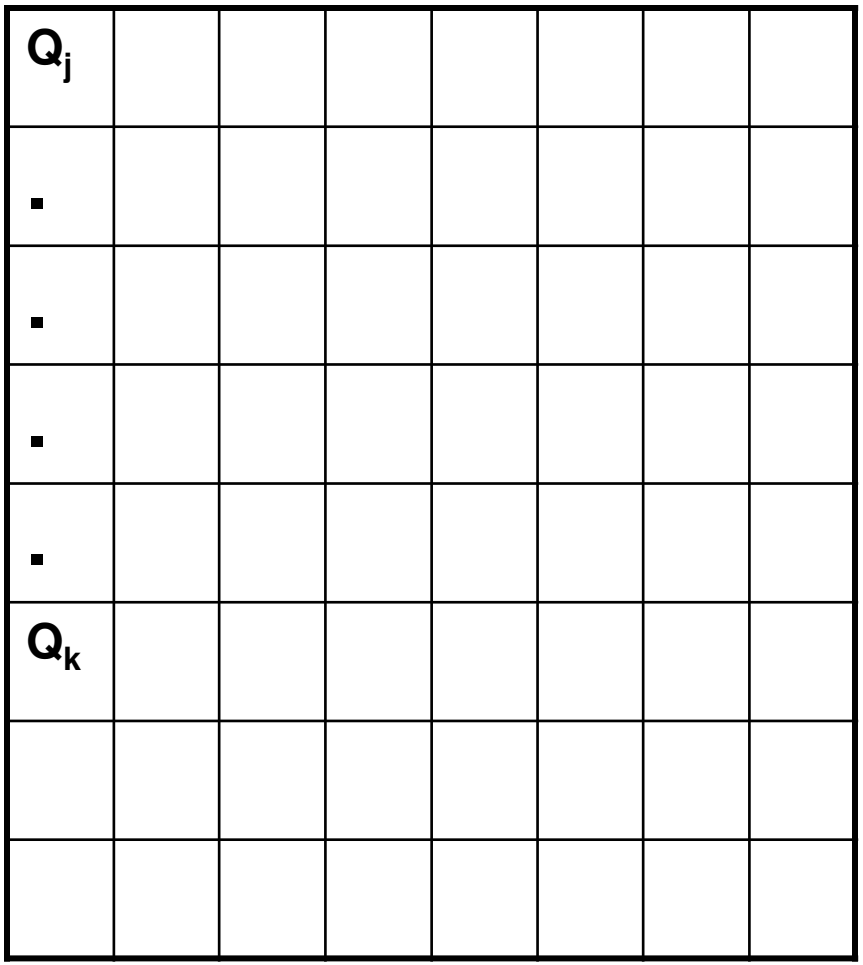
a[ ]

0	**0**
?	**1**
.	.
.	.
.	.
.	.
.	.
.	**7**

8 Queens (How can two queens hit each other)

Karoly.Bosa@jku.at

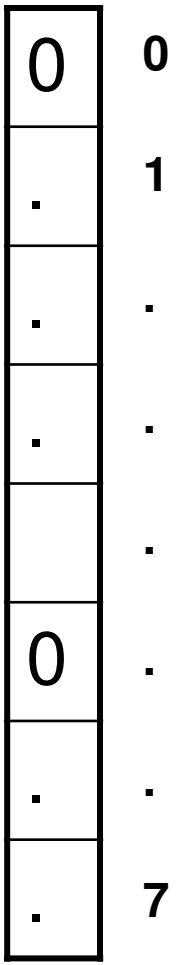
1.) They are in the same column.



We must check the following expression:

a[j] != a[k]

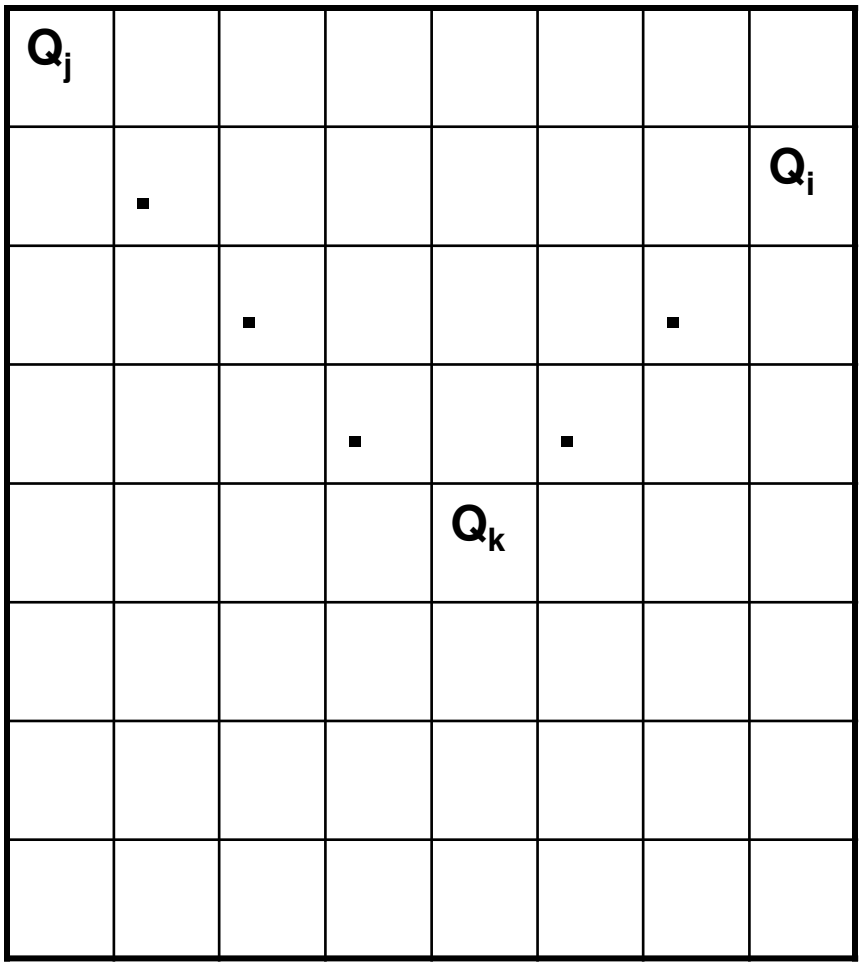
a[ ]



8 Queens (How can two queens hit each other)

Karoly.Bosa@jku.at

1.) They are in the same diagonal.



We must check the following expression:

Math.abs(a[k]-a[j])!=k-j

E.g.:

Math.abs(4-0)!=4-0
or
Math.abs(4-7)!=4-1

a[ ]

0	0
7	1
.	.
.	.
4	.
.	.
.	.
.	7

8 Queens


```
public class EightQueens {
 private static boolean findSolution(int[] a, int next) {
    …
 }

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

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

-This program will terminate after it finds the first solution.
-We use only an array with 8 elements instead of a matrix with 64 (8x8) elements.

8 Queens – methods *main, printOut*

Karoly.Bosa@jku.at

```
private static void printOut(int[] a) {
   for (int i = 0; i < a.length; i++) {
      for (int j = 0; j < 8; j++) {
        if (j!=a[i]) {
                 System.out.print(". ");
        }
        else {
                System.out.print("* ");
        }
      }
      System.out.println();
   }
}

 public static void main(String[] args) {
  int[] arrayOfRows = {-1, -1, -1, -1, -1, -1, -1, -1};
  boolean rv = findSolution(arrayOfRows, 0);
  if (rv) printOut(arrayOfRows);
 }
```

8 Queens – method *findSolution*

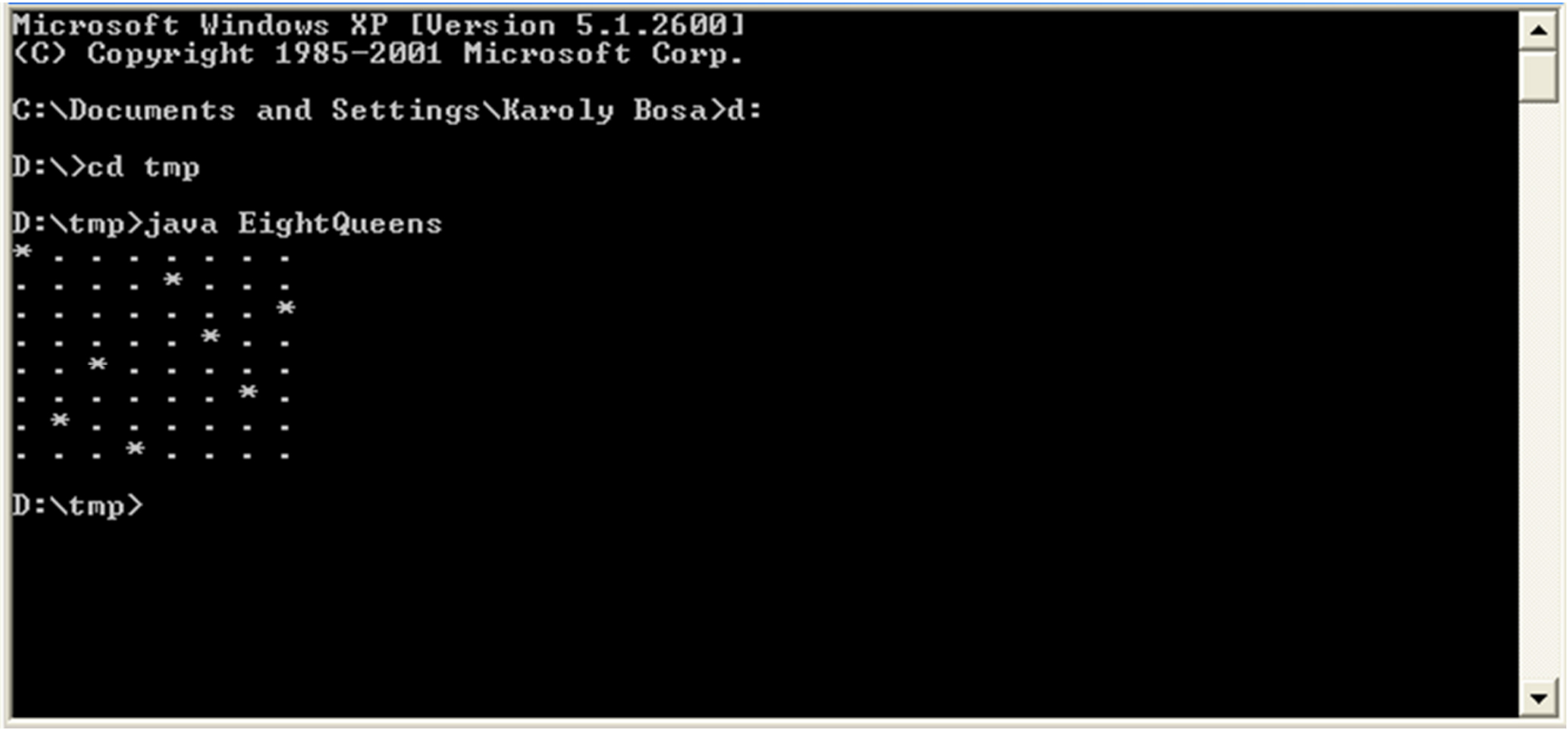
Karoly.Bosa@jku.at

```
private static boolean findSolution(int[] a, int next) {
   for (int i = 0; i < a.length ; i++) {  // i runs through the 8 position of the current queen
     a[next] = i;

     int j = 0;                              // j runs through the already placed queens
     while (j<next && a[next]!=a[j] && Math.abs(a[next]-a[j])!=next-j) {
         j++;
     }

     if (j == next) {                        //if there is not a queen hits the current one …
       if (next < a.length-1) {
          boolean rv = findSolution(a, next + 1);
          if (rv) return true;
       }
       else {                                //if we managed to place a queen into each row…
           return true;
       }
     }
   }
   return false;
}
```

8 Queens - Output



8 Queens – 2nd version

Problem: How to place 8 queens on chess-board, such that they do not capture each other.

Find all solutions with backtracking

8 Queens (2. version)


```
import java.io.IOException;

public class EightQueens2 {

 private static int numberOfSolutions = 0;

 private static void findSolution(int[] a, int next) throws IOException {
    …
 }

 private static void printOut(int[] a) throws IOException {
    …
 }

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

8 Queens (2. v.) – methods *main, printOut*

Karoly.Bosa@jku.at

```
private static void printOut(int[] a) throws IOException {
   for (int i = 0; i < a.length; i++) {
      for (int j = 0; j < 8; j++) {
        if (j!=a[i]) {
                System.out.print(". ");
        }
        else {
                System.out.print("* ");
        }
      }
      System.out.println();
   }
   System.in.read();
   System.out.println();
}

 public static void main(String[] args) throws IOException {
  int[] arrayOfRows = {-1, -1, -1, -1, -1, -1, -1, -1};
  findSolution(arrayOfRows, 0);
}
```

8 Queens (2. v.) – method *findSolution*

Karoly.Bosa@jku.at

```
private static void findSolution(int[] a, int next) throws IOException {
   for (int i = 0; i < a.length ; i++) {  // i runs through the 8 position of the current queen
     a[next] = i;

     int j = 0;                                  // j runs through the already placed queens
     while (j<next && a[next]!=a[j] && Math.abs(a[next]-a[j])!=next-j) {
       j++;
     }

     if (j == next) {                        //if there is not a queen hits the current one …
       if (next < a.length-1) {
              findSolution(a, next + 1);
       }
       else {                                  //if we managed to place a queen into each row…
              numberOfSolutions++;
              System.out.println("The " + numberOfSolutions + ". solution is:");
              printOut(a);
       }
     }
   }
}
```

8 Queens (2. version) - Output

Karoly.Bosa@jku.at