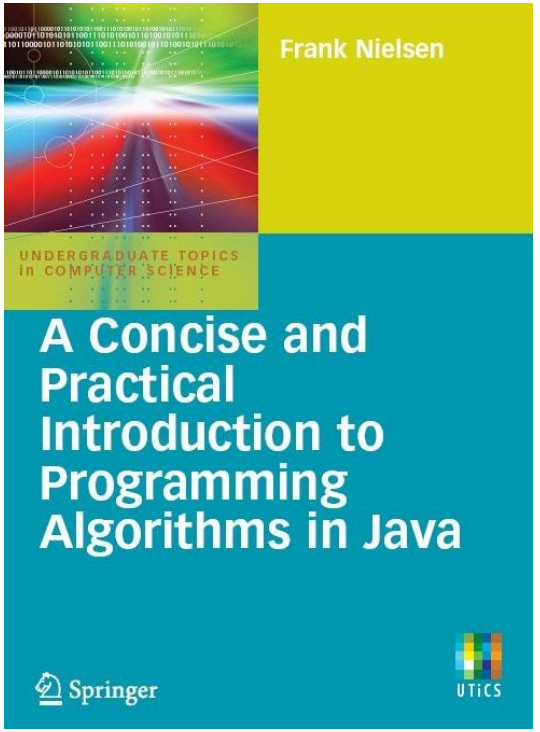
A Concise and Practical Introduction to Programming Algorithms in Java



Chapter 4: Arrays and strings

Frank NIELSEN

nielsen@lix.polytechnique.fr

Why do we need arrays?

- To handle many variables at once
- Processing many data or generating many results
- In mathematics, we are familiar with variables with indices:

$$S_n = x_1 + x_2 + \ldots + x_n = \sum_{i=1}^{n} x_i.$$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad A_{1,1} = 1,\ A_{1,2} = 2,\ \ldots,\ A_{3,2} = 8,\ A_{3,3} = 9$$

In most languages, indices start at zero (and not one).
...Just a convention

$(x_1, x_2, x_3, ...)$

$(x_0, x_1, x_2, ...)$

Declaring arrays in Java

For a given type, **TYPE[ ]** is the type of arrays storing elements of type TYPE.

- For arrays declared within the scope of functions:

- `int [ ] x;`
- `boolean [ ] prime;`
- `double [ ] coordinates;`
- `float [ ] [ ] matrix;`

- For arrays declared in the body of a class, use the keyword **static**:
  (array variables can be used by any function of the class)

- `static int [ ] x;`
- `static boolean [ ] prime;`

Building and initializing arrays

clarray.java

```
class declarray{


    static int digit [] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};

    static double x [] = {Math.PI, Math.E, 1.0, 0.0};

    static boolean prime[]={ false, true, true, true, false, true, false, true, false, false };


    static void MyFunction(int n)
    {
        int y [];

        // Allocate an array of size n
        y=new int[n];

    }


    pu
    {


}
```

Observe:

static boolean prime[ ]={ false, true, true, true, false, true, false, true, false, false };

but not

static boolean prime[10]={ false, true, true, true, false, true, false, true, false, false };

Building and initializing arrays

- Declare array with the reserved keyword **new**
- **Specify** the size of the array at built time
- Arrays can be declared and initialized **at once** too:
- `int [ ] x;`
- `x=new int [32];`
- `boolean [ ] prime = new boolean[16];`
- Arrays initialized by **enumerating** all its values:

```
int [ ] prime={2, 3, 5, 7, 11, 13, 17, 19};
```

Size of arrays

Size of arrays is given by the member function **length**:

```
prime.length;
System.out.println(prime.length);
```

Size of arrays **fixed for once**, **cannot be changed**:

```
array.length=23; // Generate an error
```

Size of arrays

declarray2.java

```
1  class declarray2{
2
3
4      public static void MyFunction(int n)
5      {
6          int array []=new int [n];
7          int i;
8
9          InformationArray(array);
10     }
11
12
13     public static void InformationArray(int [] t)
14     {
15         System.out.println("Size of array given in argument is:"+t.length);
16     }
17
18
19     public static void main (String[] args)
20     {
21
22         MyFunction(2312);
23
24         MyFunction(2008);
25
26     }
27
28
29
30 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Size of array given in argument is:2312
Size of array given in argument is:2008
Press any key to continue...
```

Index range of arrays and exceptions

Powerful mechanism of modern languages (Java, C++)

If index is out of range, an **exception** is raised on the fly:
ArrayIndexOutOfBounds

Out of range accesses **may not be detected** by the compiler:
Bug that yields termination or system crash

... However, fortunately, Java can **catch** exceptions too.

Size of arrays cannot be modified

declarray3.java

```
1  class declarray3{
2
3
4      public static void main (String[] args)
5      {
6
7
8          int x []=new int [12];
9
10         x.length=7;
11
12     }
13
14
15
16 }
```

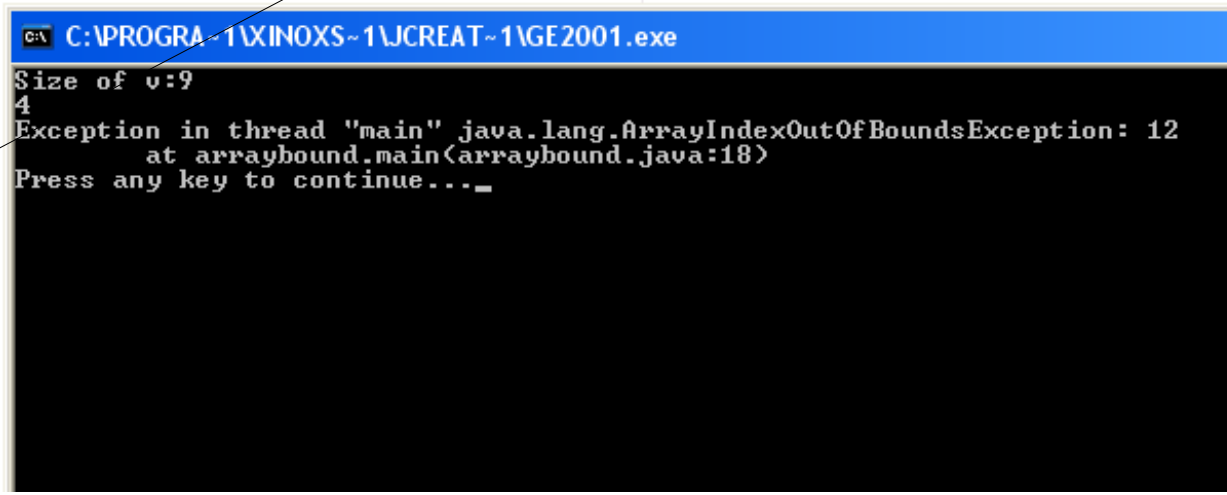
```
ild Output
--------------------Configuration: <Default>--------------------
D:\Enseignements\INF311\Lectures2008\prog-inf311.4\declarray3.java:10: cannot assign a value to final variable length
                x.length=7;
                ^
1 error

Process completed.
```

Index range of arrays and exceptions

raybound.java

```
1  class arraybound{
2
3
4      public static void main (String[] args)
5      {
6
7          int[] u = new int [16];
8
9          int [     ] v={0,1,2,3,4,5,6,7,8};
0
1
2          long l=v.length;
3
4          System.out.println("Size of v:"+l);
5
6          System.out.println(v[4]);
7
8          System.out.println(v[12]);
9
0      }
1
2
3
4  }
```

Observe the correct parsing



The concept of references

An array is allocated as a single contiguous memory block

Java is managing memory so you do not have to free it once the array is not used anymore: **garbage collector**

An array variable is, in fact, a **reference** to the array

This reference of the array is the **symbolic address** of the first element (index 0)

Thus, when we write...

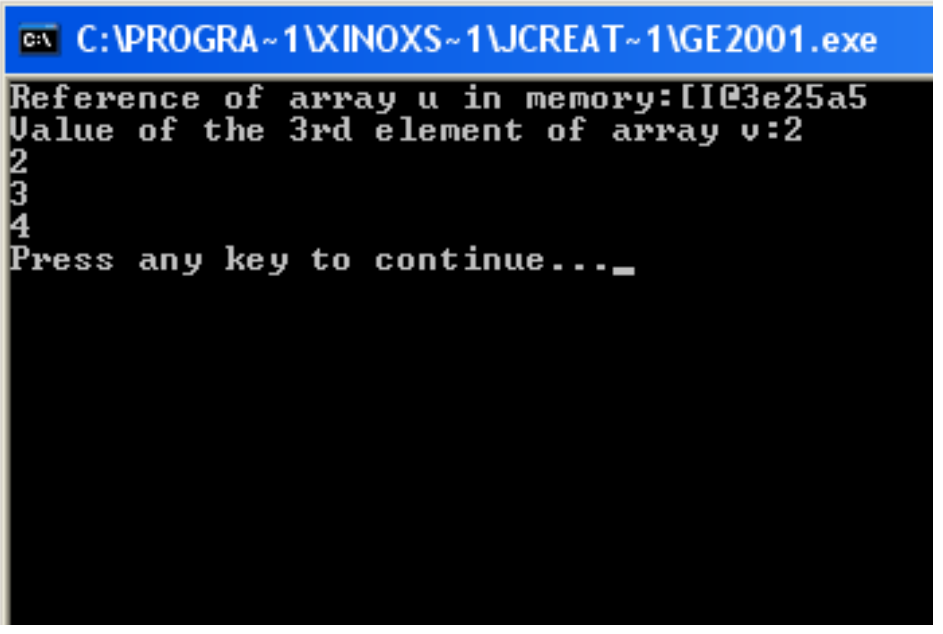
```
int [ ] v = {0, 1, 2, 3, 4};
int [ ] t =v;
t[2]++;
System.out.println(t[2]++);
```

... the elements of the array v are not copied verbatim to t. Only the reference!!!

Arrays & references

arrayref.java

```
1  class arrayref{
2
3
4      public static void main (String[] args)
5      {
6
7          int[] u = new int [5];
8
9          int [] v={0,1,2,3,4};
10
11
12         System.out.println("Reference of array u in memory:"+u);
13
14         System.out.println("Value of the 3rd element of array v:"+v[2]);
15
16         // Declare a new array
17         int [] t =v;
18
19         System.out.println(v[2]);
20         t[2]++;
21         System.out.println(v[2]);
22         v[2]++;
23         System.out.println(t[2]);
24
25     }
26
27
28
29 }
```



Functions & arrays

Functions and procedures can have arrays as arguments. (remember that array types are: TypeElement[ ])

Example: Function that returns the minimum of an array of integers

arraymin.java

```
1  class arraymin{
2
3      static int minArray(int [] t)
4      {
5          int m=t[0];
6
7          for(int i=1;i<t.length; ++i)
8              if (t[i]<m)
9                  m=t[i];
10
11                 return m;
12     }
13
14     public static void main(String[] args)
15     {
16     int [] v=new int [23];
17
18     for(int i=0;i<23;i++)
19         v[i]=25-i;
20
21         System.out.println("The minimum of the array is :"+minArray(v));
22     }
23
24 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
The minimum of the array is :3
Press any key to continue...
```

Functions & arrays

Example: Function that returns the inner product of 2 vectors (produit scalaire)

$$\langle (x_1, \ldots, x_n), (y_1, \ldots, y_n) \rangle := \sum_{i=1}^{n} x_i y_i = x_1 y_1 + \cdots + x_n y_n$$

innerproduct.java

```
1  class innerprod{
2
3      static double innerproduct(int [] x, int [] y)
4      {
5          double sum=0.0;
6
7              System.out.println("Dim of vector x:"+x.length+ " Dim of vector y:"+y.length);
8
9          for(int i=0;i<x.length; ++i)
10                 sum=sum+x[i]*y[i];
11
12                 return sum;
13     }
14
15     public static void main(String[] args)
16     {
17     int dimension=30;
18
19     int [] v1,  v2;
20
21     v1=new int[dimension];
22     v2=new int[dimension];
23
24     for(int i=0;i<dimension;i++)
25         {v1[i]=i;
26          v2[i]=i+1;
27         }
28
29
30         System.out.println("The inner product of v1 and v2 is "+innerproduct(v1,v2));
31     }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Dim of vector x:30 Dim of vector y:30
The inner product of v1 and v2 is 8990.0
Press any key to continue...
```

Array arguments in functions

A variable that has a type array is a **reference** to the array
(the memory address of the first element)

Therefore an argument of type array **does not copy**
all array elements in the memory allocated for the function,
but rather allocate a **single memory reference**:
a machine word.

```
static void MyFunction(int [ ] x)
MyFunction(v);
```

Only the *reference of v* is copied to the memory allocated for the function MyFunction.

Array arguments in functions

Thus we can modify the inside a function the contents of the array: the values of the elements of the array.

modifyarray.java

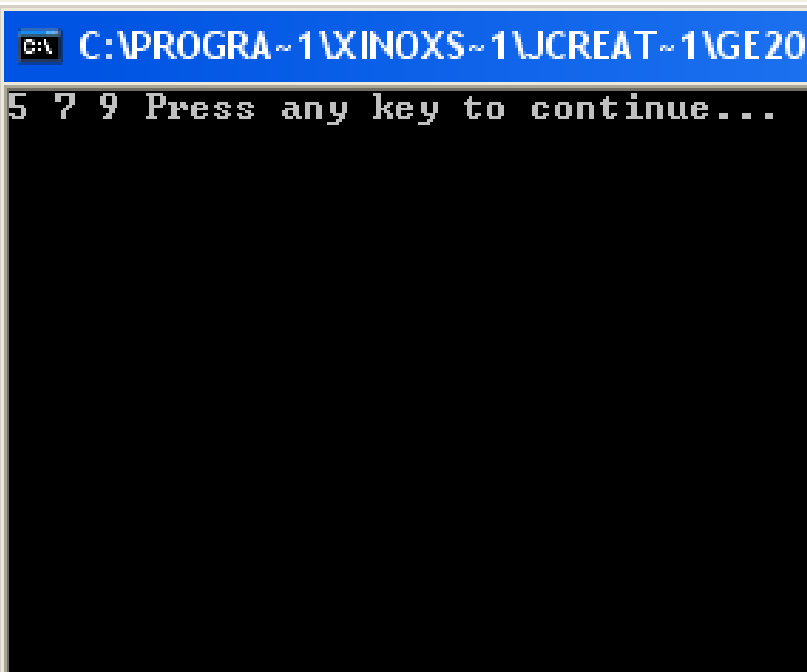
```
1  class modifyarray{
2
3  static void swap(int [] t, int i, int j)
4  {
5      int tmp;
6
7      tmp=t[i];
8      t[i]=t[j];
9      t[j]=tmp;
10 }
11
12 static void DisplayArray(int [] x)
13 {
14     for(int i=0;i<x.length;i++)
15     System.out.print(x[i]+" ");
16
17     System.out.println();
18 }
19
20     public static void main(String[] args)
21     {
22
23         System.out.println("This program shows how to modify inside a function the contents of an array");
24
25         int [] t={1,2,3,4,5,6,7,8,9};
26
27         DisplayArray(t);
28
29         swap(t,2,3);
30
31         DisplayArray(t);
32     }
33
34 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
This program shows how to modify inside a function the contents of an array
1 2 3 4 5 6 7 8 9
1 2 4 3 5 6 7 8 9
Press any key to continue...
```

Functions returning an array

addvector.java

```
1  class addvector{
2
3      static int [] addvector(int [] u, int [] v)
4      {
5          int[] result;
6
7          result=new int[u.length];
8
9          for(int i=0;i<u.length;i++)
10             result[i]=u[i]+v[i];
11
12             return result;
13     }
14
15
16     public static void main(String[] args)
17     {
18         int [] x={1, 2, 3};
19         int [] y={4, 5, 6};
20
21         int [] z= addvector(x,y);
22
23     for(int i=0;i<z.length;i++)
24         System.out.print(z[i]+" ");
25     }
26
27 }
```



Arrays of arrays...

So far, we described **linear array** (1D).
What about matrices (2D arrays)?

A bidimensional array (n,m) consists of **n lines**,
each of which is an array of **m elements**

```
int [ ] [ ] matrix;
matrix=new int[n][m];
```

By default, at initialization, the array is **filled up with zero**
Change the contents of 2D arrays using 2 **nested loops**:

```
for(int i=0; i<n; i++)
  for(int j=0; j<m;j++)
      matrix[i][j]=i*j+1;
```

2D Arrays: Matrix x vector product

matrixvector.java

```
1  class matrixvector{
2
3      static int [] MultiplyMatrixVector(int [][] mat, int [] v)
4      {
5          int[] result;
6
7          result=new int[mat.length];
8
9          for(int i=0;i<result.length;i++)
10         {
11             result[i]=0;
12
13             for(int j=0;j<v.length;j++)
14
15             result[i]+= mat[i][j]*v[j];
16         }
17             return result;
18     }
19
20
21     public static void main(String[] args)
22     {
23         int [][] M={{1, 2, 3}, {4,5,6}, {7,8,9}};
24         int [] v={1,2,3};
25
26         int [] z= MultiplyMatrixVector(M,v);
27
28     for(int i=0;i<z.length;i++)
29         System.out.print(z[i]+" ");
30     }
31
32 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
14 32 50 Press any key to continue...
```

Dichotomic search

Also called binary search algorithm

Assume we are given a **sorted array** of size n:

$$array[0] < array[1] < ... < array[n-1]$$

and a **query key** p

Seek whether there is an element in the array that has value p

That is, give a function that return the **index** of the element in the array with value p, or that returns -1 otherwise.

Dichotomic search: Think recursion!

- Start with a search interval [left, right] with left=0 and right=n-1
- Let m denote the **middle** of this interval: m=(left+right)/2
- If array[m]=p then we are done, and we return m;
- If array[m] <a, then if the solution exists it is in [m+1,right]
- If array[m]>a, then if the solution exists it is in [left,m+1]
- The search algorithm terminates if left>right, we return -1;

Dichotomic search: Recursion

dichotomysearch.java

```
1  class dichotomysearch{
2
3
4      static int Dichotomy(int [] array, int left, int right, int key)
5      {
6
7          if (left>right) return -1;
8
9          int m=(left+right)/2;
10
11         if (array[m]==key) return m;
12         else
13         {
14             if (array[m]<key) return Dichotomy(array,m+1, right, key);
15                 else  return Dichotomy(array,left,m-1, key);
16         }
17
18     }
19
20
21     static int DichotomicSearch(int [] array, int key)
22     {
23         return Dichotomy(array,0,array.length-1, key);
24     }
25
26     public static void main (String[] args)
27     {
28     int [] v={1,6,9 ,12 ,45, 67, 76, 80, 95};
29
30         System.out.println("Seeking for element 6: Position "+DichotomicSearch(v,6));
31         System.out.println("Seeking for element 80: Position "+DichotomicSearch(v,80));
32         System.out.println("Seeking for element 33: Position "+DichotomicSearch(v,33));
33     }
34
35 }
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Seeking for element 6: Position 1
Seeking for element 80: Position 7
Seeking for element 33: Position -1
Press any key to continue...
```

Strings: Basic objects in Java

- A string of character is **an object** with type **String**

- A variable of type String is a **reference** on that object:

```
String school= "Ecole Polytechnique";
String vars=school;
```

- Once built, a string object **cannot** be modified

- Beware: use only for moderate length strings,
  otherwise use the **class StringBuffer**

Class String: Some methods

A method is a function or procedure on an object class

Method Length() : gives the number of characters

```
String s= ''anticonstitutionnellement'';
System.out.println(s.length());
```

Method equals():

s1.equals(s2): Predicate that returns true **if and only if** the two strings s1 and s2 are made of the same sequence of characters.

```
String s1=''Poincare'';
String s2=TC.lireMotSuivant();
System.out.println(s1.equals(s2));
```

Beware: s1==s2 is different!

It compares the **reference** of the strings.

(Physical versus logical equality test)

Class String in action...

ngmethod.java

```
class stringmethod{


    public static void main(String[] args)
    {
        String name="Ecole Polytechnique";
        String promotion="2008";

        String fullname=name+" "+promotion;

        System.out.println(fullname);
        System.out.println("Length:"+fullname.length());


        System.out.println("Type a sentence so that I check whether it is Poincare or not");
        String pattern="Poincare";
        String query=TC.lireMotSuivant();

        System.out.println(pattern.equals(query));

    }


}
```

```
C:\PROGRA~1\XINOXS~1\JCREAT~1\GE2001.exe
Ecole Polytechnique 2008
Length:24
Type a sentence so that I check whether it is Poincare or not
Poincare
true
Press any key to continue...
```

Class String: More methods

Method **charAt()**:
s.charAt(i) gives the character at the (i+1)th position in string s.

```
String s= ''3.14159265'';
System.out.println(s.charAt(1));
```

Method **compareTo()**:
u.compareTo(v) compares lexicographically the strings u with v.

```
String u=''lien'', v=''lit'', w=''litterie'';
System.out.println(u.compareTo(v));
System.out.println(v.compareTo(w));
```

Class String: More methods

stringmethod2.java

```
1  class stringmethod2
2  {
3
4      public static void main(String[] args)
5      {
6          String s="3.141559";
7
8          System.out.println(s.charAt(1));
9
10         String u="lien", v="lit", w="litterie";
11         System.out.println(u.compareTo(v));
12         System.out.println(v.compareTo(w));
13
14     }
15
16 }
```

```
C:\PROGRA~1\XINOXS~1\JCR
.
-15
-5
Press any key to continue
```

Demystifying the main function

```
class ClassName
{
   public static void main(String[ ] args)
   {
   ...
   }
}
```

Function main has an array of string of characters as arguments
These strings are stored in args[0], args[1], ...
... when calling java main s0 s1 s2 s3

Use Integer.parseInt() to convert a string into an integer

```
D:\J>javac main.java

D:\J>java main a small test to parse as a command line
0:a
1:small
2:test
3:to
4:parse
5:as
6:a
7:command
8:line

D:\J>
```

Parsing arguments in the main function

parsingarg.java

```
1  class parsingarg{
2
3
4
5      public static void main(String[] args)
6      {
7
8          String first=args[0];
9
10         for(int i=1; i<args.length;i++)
11         if (first.compareTo(args[i])>0)
12         first=args[i];
13
14         System.out.println("Lexicographically maximum string is:"+first);
15     }
16
17 }
```

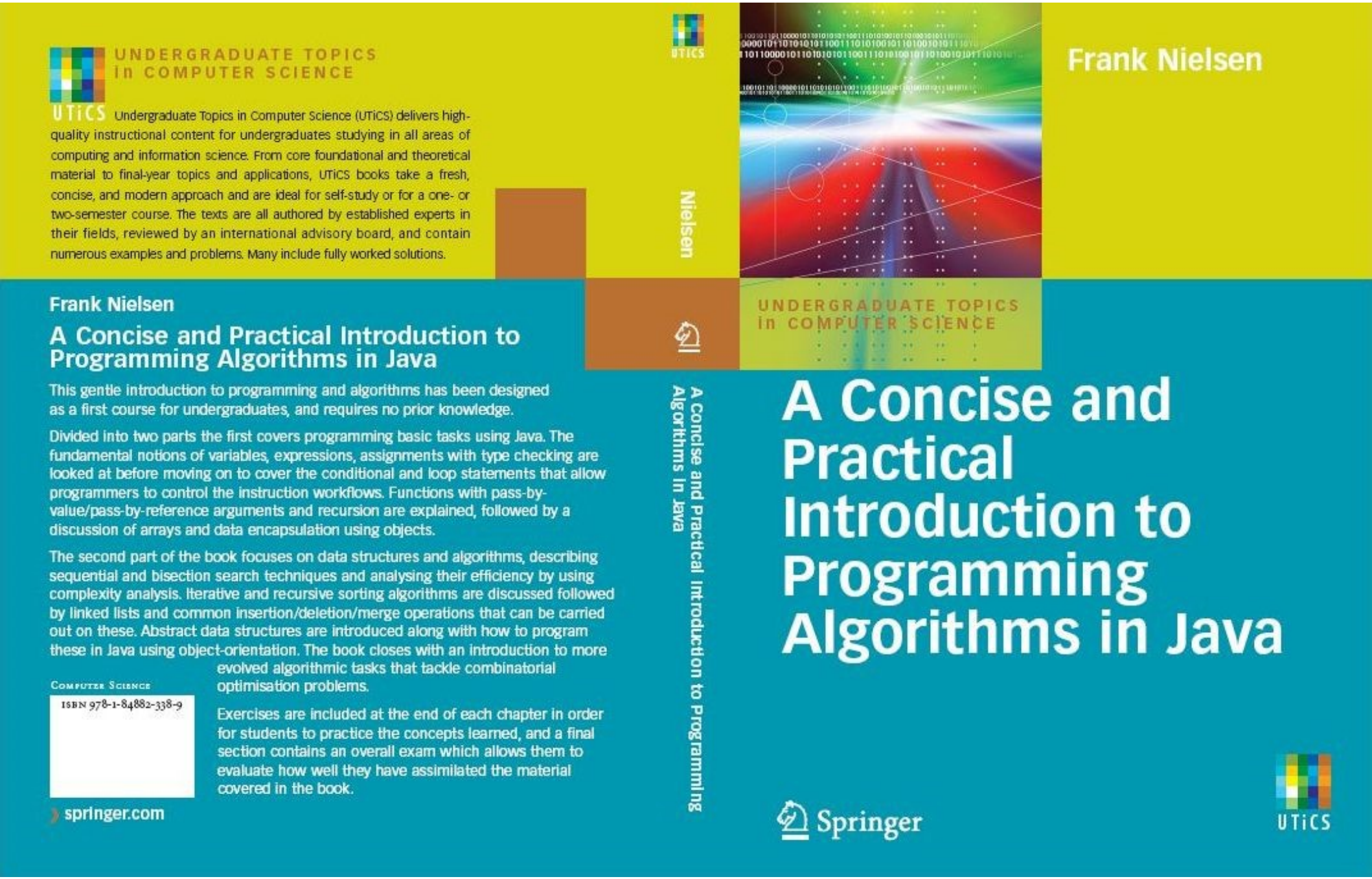
```
D:\J>java parsingarg lit lien litterie
Lexicographically maximum string is:lien

D:\J>
```

Parsing arguments in the main functio

parsingarg2.java

```
1  class parsingarg2{
2
3
4
5      public static void main(String[] args)
6      {
7
8          int first=0;
9
10         for(int i=1; i<args.length;i++)
11         if  (Integer.parseInt(args[first])>Integer.parseInt(args[i]))
12         first=i;
13
14         System.out.println("Location of minimum argument:"+first);
15     }
16
17 }
```

```
D:\J>javac parsingarg2.java

D:\J>java parsingarg2 9 4 6 2 6 4 1 3 5 4 6
Location of minimum argument:6

D:\J>_
```


UNDERGRADUATE TOPICS
in COMPUTER SCIENCE
UTiCS
Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.
Frank Nielsen
A Concise and Practical Introduction to Programming Algorithms in Java
This gentle introduction to programming and algorithms has been designed as a first course for undergraduates, and requires no prior knowledge.
Divided into two parts the first covers programming basic tasks using Java. The fundamental notions of variables, expressions, assignments with type checking are looked at before moving on to cover the conditional and loop statements that allow programmers to control the instruction workflows. Functions with pass-by-value/pass-by-reference arguments and recursion are explained, followed by a discussion of arrays and data encapsulation using objects.
The second part of the book focuses on data structures and algorithms, describing sequential and bisection search techniques and analysing their efficiency by using complexity analysis. Iterative and recursive sorting algorithms are discussed followed by linked lists and common insertion/deletion/merge operations that can be carried out on these. Abstract data structures are introduced along with how to program these in Java using object-orientation. The book closes with an introduction to more evolved algorithmic tasks that tackle combinatorial optimisation problems.
Exercises are included at the end of each chapter in order for students to practice the concepts learned, and a final section contains an overall exam which allows them to evaluate how well they have assimilated the material covered in the book.
Computer Science
ISBN 978-1-84882-338-9
springer.com
UTiCS
Nielsen
A Concise and Practical Introduction to Programming Algorithms in Java
Frank Nielsen
UNDERGRADUATE TOPICS
in COMPUTER SCIENCE
A Concise and Practical Introduction to Programming Algorithms in Java
Springer
UTiCS