4 Non-primitive data types in Java

- Strings
- Arrays
- StringTokenizer
- StringBuffer

## **String** from **java.lang.String**

- is *not* a basic type like **int**, **float** etc.
- is a class (**class**)
- belongs to the package **java.lang** (i.e., no import required)
- can sometimes be used like a basic type

Declaration / copy:

```
1  String str1;                    //declaration of a variable
2
3  String str2 = "a String";   //declaration with initialization
4
5  str1 = str2;                    //copy of the reference variable
6
7  char text[] = {'a', 'b', 'c'};
8  str1 = new String(text);    //redeclaration with initialization
```

'**str1**', '**str2**' are reference variables that can only refer to objects from the class **String** (see later...).

- Methods in String are NOT **`static`**.
- We thus need to refer the object on which the method should be applied.

Example: **`public int length ( )`**

```
1  public class K4B01E_StringLength {
2
3   public static void main (String[] args) {
4
5    String st1 = "a String";
6    String st2 = "another String";
7
8    System.out.println("\"" + st1 + "\": length " + st1.length() );
9    System.out.println("\"" + st2 + "\": length " + st2.length() );
10   st2 = st1;
11   System.out.println("\"" + st2 + "\": length " + st2.length() );
12  }
13 }
```

## Extraction of single characters and substrings

Consider `String aString = "This is a String";`

| T | h | i | s | | i | s | | a | | S | t | r | i | n | g |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

- `public char charAt (int index)`
  yields character at position **`index`** (from **`0`** to **`length-1`**)

  `aString.charAt (3);` $\rightsquigarrow$ '**`s`**'

- `public String substring (int beginIndex)`
  yields substring from **`beginIndex`** to end of String

  `aString.substring (10);` $\rightsquigarrow$ "**`String`**"

- `public String substring (int beginIndex, int endIndex)`
  yields substring from **`beginIndex`** to **`endIndex-1`**

  `aString.substring (0, 4);` $\rightsquigarrow$ "**`This`**"

| T | h | i | s | | i | s | | a | | S | t | r | i | n | g |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

- `public int indexOf (char ch )`

  yields first position of **`ch`** (forward search starting at 0)

  ```
  index = aString.indexOf ( 'i' );          ⇝  index = 2
  ```

- `public int indexOf (char ch , int fromIndex)`

  yields next position of **`ch`** (forward search starting at **`fromIndex`**)

  ```
  index = aString.indexOf ( 'i' , index + 1 );    ⇝  index = 5
  ```

- `public int lastIndexOf (char ch )`

  yields last position of **`ch`** (backward search starting at **`length-1`**)

  ```
  index = aString.lastIndexOf ( 'i' );        ⇝  index = 13
  ```

- `public int lastIndexOf (char ch , int fromIndex)`

  yields next position of **`ch`** (backward search starting at **`fromIndex`**)

  ```
  index = aString.lastIndexOf ( 'i' , index-1);    ⇝  index = 5
  ```

# Localizing substrings in a String

| T | h | i | s | | i | s | | a | | S | t | r | i | n | g |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

- `public int indexOf (String str)`

  yields first position of **`str`** (forward search starting at 0)

  ```
  index = aString.indexOf ( "is" );          ⇝  index = 2
  ```

- `public int indexOf (String str, int fromIndex)`

  yields next position of **`str`** (forward search starting at **`fromIndex`**)

  ```
  index = aString.indexOf ( "is" , index + 1 );   ⇝  index = 5
  ```

- `public int lastIndexOf (String str)`

  yields last position of **`str`** (backward search starting at **`length-1`**)

  ```
  index = aString.lastIndexOf ( "his" );       ⇝  index = 1
  ```

- `public int lastIndexOf (String str, int fromIndex)`

  yields next position of **`str`** (backward search starting at **`fromIndex`**)

  ```
  index = aString.lastIndexOf ( "his" , index-1);  ⇝  index = -1
  ```

## Manipulation of Strings

Functions with String as return type:

- The input String is **not** modified,
- the manipulations result in new Strings
- Strings are **immutable**, i.e., they cannot be modified

```
1 public String replace (char oldChar, char newChar)
2 public String toLowerCase ( )
3 public String toUpperCase ( )
4 public String trim ( )
5 public String concat (String str)
```

Example with **`String aString = "a String"`**

```
1  aString.replace ('i', 'u')  ⇝ "a Strung"
2
3  aString.toLowerCase ()  ⇝ "a string"
4
5  aString.toUpperCase ()  ⇝ "A STRING"
6
7  "␣␣␣a String    ".trim ()  ⇝ "a String"
8
9  "extend ".concat (aString)  ⇝ "extend a String"
10 aString.concat (" is extended")  ⇝ "a String is extended"
```

# String comparisons

```
1 public boolean startsWith (String prefix)
2 public boolean startsWith (String prefix, int index)
3 public boolean endsWith (String suffix)
4 public boolean regionMatches (int cindex,
5              String str, int strindex, int size)
6 public boolean regionMatches (boolean case, int cindex,
7           String str, int strindex, int size)
```

## Example with `String aString = "This is a String"`

```
1 aString.startsWith ("this")                          ↪ false
2 aString.startsWith ("is", 5)                         ↪ true
3 aString.endsWith ("ing")                             ↪ true
4
5 aString.regionMatches (13, "KRINGEL", 2, 3)          ↪ false
6 aString.regionMatches (true, 13, "KRINGEL", 2, 3)    ↪ true
7                                                  // ignore case
```

## String equality

```
1 public boolean equals (String anotherString)
2 public boolean equalsIgnoreCase (String anotherString)
```

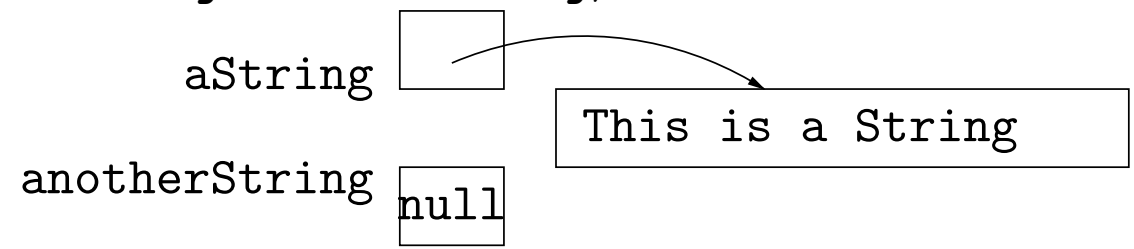
Example with **`String aString = "This is a String"`**

```
1  aString.equals ("this is a string")             ⇝ false
2  aString.equalsIgnoreCase ("this is a string")   ⇝ true
3  "This is a String".equals(aString)              ⇝ true
```

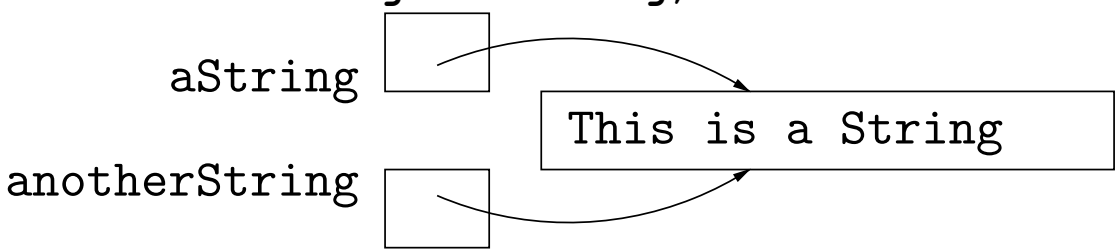
many more methods for **`String`**: see the Java documentation

# Reference variables for Strings

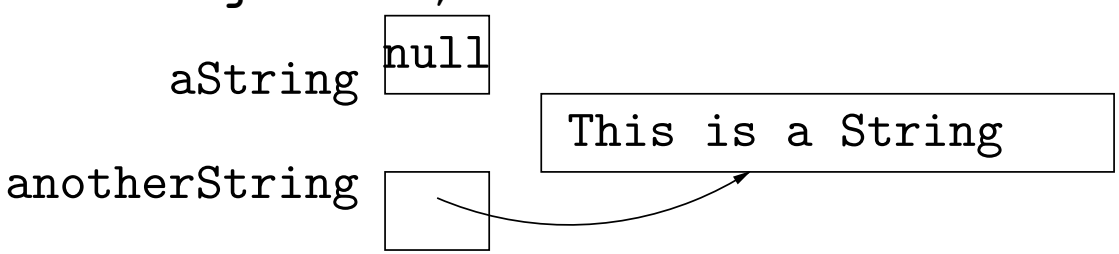
- `String aString = "This is a String";`
  `String anotherString;`




- `anotherString = aString;`




- `aString = null;`



- Reference variables do not store the actual data, but only a reference to the data.
- A comparison of two reference values (with “==”) only compares the references, not the referenced data.
- An assignment between reference variables only copies the reference, not the actual data.
  (Since Strings are immutable, copying a String would not make much sense anyway...)

## Preview of the concept of classes in Java

```
1       str = new String ( );
2       str = new String ( "a String");
3       str = new String ( {'a', 'b', 'c'} );
```

- **`String()`**, **`String(String str)`**, **`String(char[] chArray)`** are *constructors* of the class **`java.lang.String`**.
- **`str = new String ( {´a´, ´b´, ´c´} )`** causes:
  - generation of a new *object* and initialization with the values from the char array, i.e., as **`"abc"`**.
  - assignment of a reference to the object to the variable **`str`**.

Attention:

- Do **not** use "**==**" to compare Strings!
- This compares only the references, not the Strings!

```
1  public class K4B02E_StringEqual {
2    public static void main(String[] args) {
3      String g = "Hello", h = "Hello";
4      String i = "Hello" + "";
5      String j = h + "";
6
7      System.out.println( "1." + g.equals(h) );
8      System.out.println( "2." + g.equals(i) );
9      System.out.println( "3." + g.equals(j) );
10     System.out.println( "4." +  ( g == h ) );
11     System.out.println( "5." +  ( g == i ) );
12     System.out.println( "6." +  ( g == j ) );
13     System.out.println( "7." +    g == h   );
14   }
15 }
```

- 1./2./3.: content, i.e, `true`, since same content
- 4./5.: references, `true`, created at compilation time ("String pool")
- 6.: references, `false`, since not in the "String pool"
- 7.: references, but '+' stronger than '=='

## **Test for palindrome** as an example for String manipulations:

```
1  public class K4B03E_Palindrom {
2  /*  tests if a given string is a palindrome  */
3
4   public static void main(String[] args) {
5
6     String str = System.console().readLine
7               ("input test string: ");
8
9     int left, right;
10    for ( left = 0, right = str.length () - 1;
11          left < right;
12          left++, right --) {
13       if ( str.charAt (left) != str.charAt (right) ) break;
14    }
15
16    System.out.println("palindrome " +  (left >= right) );
17   }
18 }
```

## **Reversing** as an example for String manipulations:

```
1  public class K4B04E_Reverse {
2  /*  reverses the input string    */
3
4   public static void main(String[] args) {
5
6     String str = System.console().readLine("input string: ");
7
8     String reverse="";
9
10    for  (int index =0; index < str.length(); index++ ) {
11        reverse = str.charAt(index) + reverse;
12    }
13
14    System.out.println( str + "␣reversed " + reverse );
15   }
16 }
```

With the (overloaded) static method **`valueof (...)`** it is possible to convert objects of many classes to Strings:

```
1  public class K4B05E_StringValueOf {
2     public static void main( String args[] ) {
3        char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
4        boolean booleanValue = true;
5        char characterValue = 'Z';
6        int integerValue = 7;
7        long longValue = 10000000L;
8        float floatValue = 2.5f;
9        double doubleValue = 33.333;
10
11       System.out.println(
12           "char␣array:␣" + String.valueOf( charArray ) +
13         "\nboolean:␣␣␣␣" + String.valueOf( booleanValue ) +
14         "\nchar:␣␣␣␣␣␣␣" + String.valueOf( characterValue ) +
15         "\nint:␣␣␣␣␣␣␣␣" + String.valueOf( integerValue ) +
16         "\nlong:␣␣␣␣␣␣␣" + String.valueOf( longValue ) +
17         "\nfloat:␣␣␣␣␣␣" + String.valueOf( floatValue ) +
18         "\ndouble:␣␣␣␣␣" + String.valueOf( doubleValue ) );
19  }
20 }
```

(this uses **`toString()`** of the corresponding classes)

4 Non-primitive data types in Java

- Strings
- Arrays
- StringTokenizer
- StringBuffer

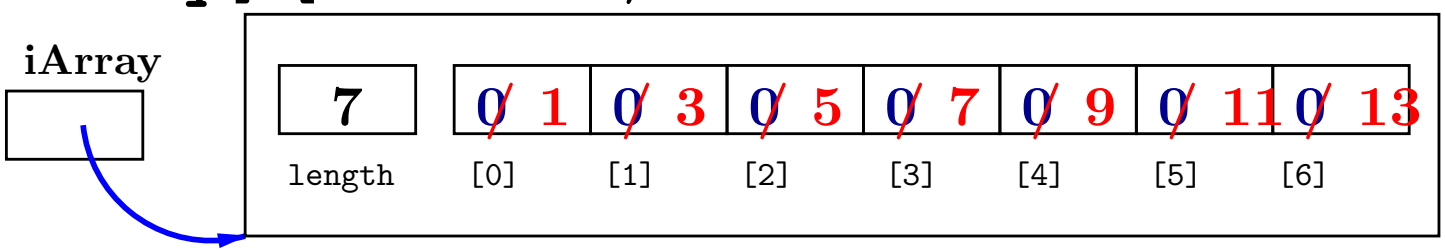
- Arrays have existed already in the first high-level programming languages to represent vectors, matrices, etc.
- Usually an array is an aggregation of multiple data elements of the same type to a data structure.
- In Java an array is a group of variables or reference variables of the same type.
- In Java this aggregation can be repeated over multiple steps.
- In other programming languages, multi-dimensional arrays form a unit (in the memory of the computer)
  Java always uses multiple levels (arrays of arrays of . . . )
- Once an array was initialized in Java, its size cannot be changed, the number of its elements is fixed.
  (but: the elements can be reference variables...).

```
1  public class K4B06E_Days {
2     public static void main (String[] args) {
3
4  //declaration and initialization of an array of dimension 1
5  //name: week,
6  //type of elements: String
7  //number of elements: 7
8
9        String [] week  = { "Monday", "Tuesday", "Wednesday",
10                           "Thursday", "Friday",
11                           "Saturday", "Sunday"};
12
13 // the indexes of the array elements run from 0 to 6
14 // (from 0 to week.length-1)
15
16       System.out.println( "Days of the week:" );
17       for (int i = 0; i < week.length; i++)
18          System.out.println("␣week [" + i + "]␣=␣" + week[i]);
19 // week [i]:  access the ith array element
20
21    }
22 }
```

- The indexes for the elements of an array **`anArray`** run from `0` to **`anArray.length-1`**.
- If an index value $<$ `0` or $\geq$ **`length`** is accessed, an **`ArrayIndexOutOfBoundsException`** is generated.
- An index value must be a non-negative integer or an integer expression, e.g,
  **`anArray [ j++ + a[0] ]`**or
  **`anArray [(int)Math.pow(2,2)]`**

Arrays are used in three steps:

- *declaration of the variable* (including assigment of its type)
  Ex: `int [] iArray;`
  `iArray` is a variable referencing an array with elements of type `int`.
- *allocation of memory & initialization*
  Ex: `iArray = new int [7];`
  allocation of memory for 7 elements of type `int`
  (automatically initialized with value 0)
  and a memory cell for the array size.
- *value assignment*
  Ex: `for (int i=0; i<iArray.length; i++)`
  `iArray[i] = 2*i+1;` with values:



- When memory is allocated, the array is already filled with default values.
- Examples for default values:
  - `int [] intArray = new int [4];`
    `↝ 0 0 0 0`
  - `float [] floatArray = new float [4];`
    `↝ 0.0 0.0 0.0 0.0`
  - `boolean [] booleanArray = new boolean [4];`
    `↝ false false false false`
  - `char [] charArray = new char [4];`
    `↝ 0x0000 0x0000 0x0000 0x0000`
  - `String [] StringArray = new String [4];`
    `↝ null null null null`

# Example: arrays as parameters

Arrays are passed using *call by reference-value*:

```
1  public class K4B07E_TestFeld {
2
3     static void showArray (int [] anArray) {
4        for (int i = 0; i < anArray.length; i++)
5           System.out.println(i + "␣:␣" + anArray [i]);
6        System.out.println();
7     }
8
9     public static void main (String[] args) {
10       int [] testArray;
11
12       testArray = new int [5];
13       for (int i = 0; i < 5; i++) testArray [i] = i + 1;
14       showArray (testArray);
15
16       testArray = new int [] { 11, 22, 33, 44 };
17       showArray (testArray);
18    }
19 }
```

# Example: evaluation of an `int` expression via the command line:

```
 1 public class  K4B08E_CLArgs {
 2    public static void main( String args[] ) {
 3       int  result;
 4       int  arg1 = Integer.parseInt ( args [ 0 ] );
 5       char arg2 = args[1].charAt(0);
 6       int  arg3 = Integer.parseInt ( args [ 2 ] );
 7       switch (arg2) {
 8          case '+': result = arg1 + arg3; break;
 9          case '-': result = arg1 - arg3; break;
10          case '*': result = arg1 * arg3; break;
11          default : result = 0;
12       }
13       System.out.println(
14            arg1 + "␣" + arg2 + "␣" + arg3 + "␣=␣" + result);
15    }
16 }
```

**`java K4B08E_CLArgs 77 + 23`** $\rightsquigarrow$ **100**

**`java K4B08E_CLArgs 129 - 32`** $\rightsquigarrow$ **97**

**`java K4B08E_CLArgs 32 "*" 32`** $\rightsquigarrow$ **1024**

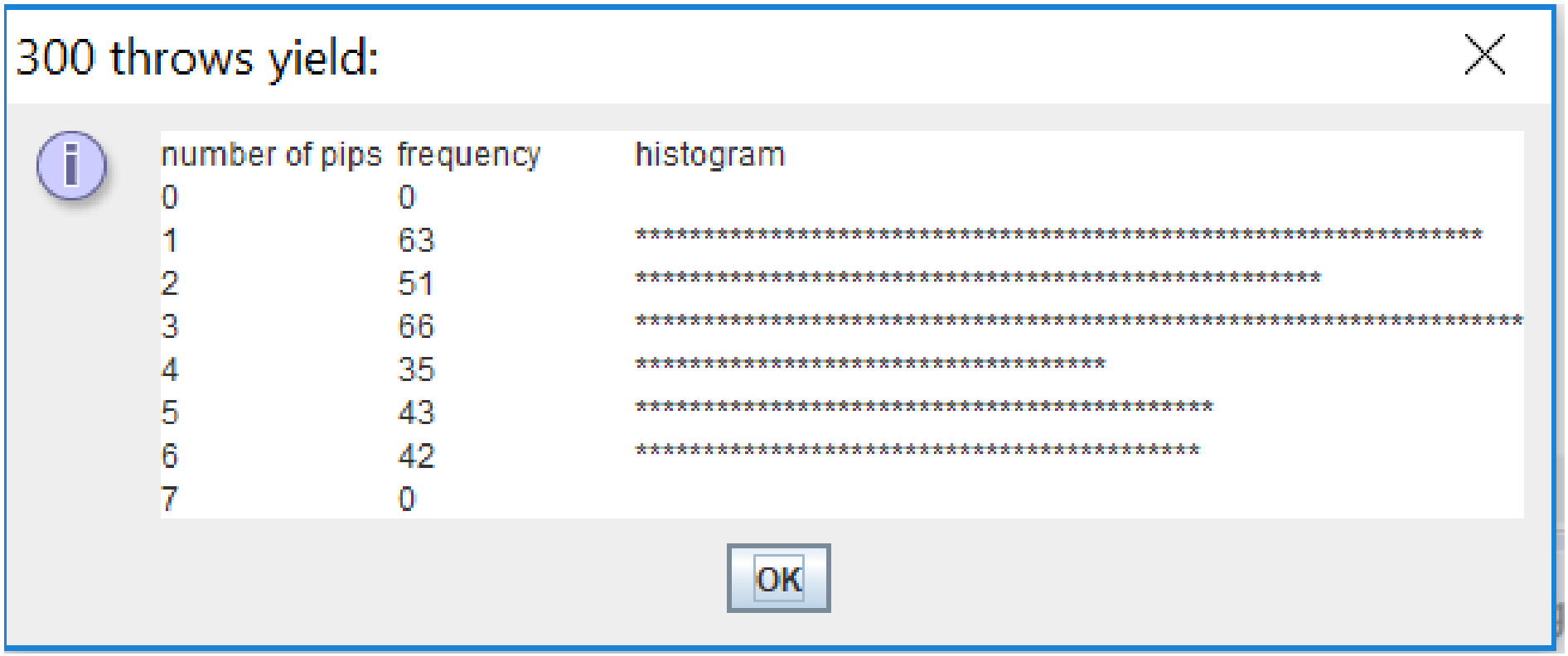
# Example: sum of the command line parameters

```
 1 public class  K4B09E_CLSum {
 2    public static void main( String args[] ) {
 3
 4       int sum = 0, i;
 5
 6       for (i = 0; i < args.length; i++)
 7          sum = sum + Integer.parseInt ( args [i] );
 8
 9       System.out.println(
10             "The sum of the " + i + "␣elements is: " + sum);
11    }
12 }
```

**java K4B09E_CLSum 234 345 456 567 678 ⇝ 2280**

# Example: distribution of the number of pips of a die

```
1  import javax.swing.*;
2  public class K4B10E_Dice {
3    public static void main( String args[] ) {
4      int frequency[] = new int[ 8 ];
5
6      for ( int roll = 1; roll <= 300; roll++ )
7         frequency [ 1 + (int) ( Math.random() * 6 ) ] ++;
8
9
10     String output = "number of pips\tfrequency\thistogram";
11     for ( int side = 0; side < frequency.length; side++ ) {
12         output += "\n" + side + "\t" + frequency[side] + "\t";
13         for ( int k = 0; k < frequency[ side ]; k++ )
14            output += "*";
15     }
16
17     JTextArea outputArea = new JTextArea();
18     outputArea.setText( output );
19     JOptionPane.showMessageDialog( null, outputArea,
20       "300 rolls yield:", JOptionPane.INFORMATION_MESSAGE );
21   }
22 }
```


300 throws yield:
number of pips frequency histogram
0 0
1 63 **********
2 51 *********
3 66 **********
4 35 ******
5 43 *******
6 42 *******
7 0
OK

# Example: sorting an array of type `char[]`

```
1  import java.util.Scanner;
2  public class K4B11E_BubbleSort {
3    public static void main(String[] args) {
4
5      Scanner sc = new Scanner(System.in);
6      System.out.print("Test string: ");
7      String ts = sc.nextLine();
8      char [] chArray = ts.toCharArray();
9
10     System.out.println ( 0 + ": " + new String(chArray) );
11
12     for (int i = 1; i < chArray.length; i++){
13       for (int j = 0; j < chArray.length - i; j++)
14         if ( chFeld [j] > chArray [j+1]) {
15           char help = chArray [j];
16           chArray [j] = chArray [j+1];
17           chArray [j+1] = help;
18         }
19       System.out.println ( i + ": " + new String(chArray) );
20     }
21   }
22 }
```

# Exemplary executions:

```
1     for (int i = 1; i < chArray.length; i++){
2       for (int j = 0; j < chArray.length - i; j++)
3         if ( chArray [j] > chArray [j+1]) {
4           char help = chArray [j];
5           chArray [j] = chArray [j+1];
6           chArray [j+1] = help;
7         }
8       System.out.println ( i + ": " + new String(chArray) );
9     }
```

**Test string: "one_String"**

```
0: one_String
1: ne_Soringt
2: e_Snoingrt
3: _Seningort
4: S_eingnort
5: S_eignnort
6: S_eginnort
7: S_eginnort
8: S_eginnort
9: S_eginnort
```

**Test string: "9876543210"**

```
0: 9876543210
1: 8765432109
2: 7654321089
3: 6543210789
4: 5432106789
5: 4321056789
6: 3210456789
7: 2103456789
8: 1023456789
9: 0123456789
```

# Sorting of an array in a method, parameter passing with '*call by reference-value*'

```
1  public class K4B12E_BubbleSort2 {
2
3     public static void sort ( int [] a) {
4       for (int i = 1; i < a.length; i++)
5           for (int j = 0; j < a.length - i; j++)
6              if ( a[j] > a[j+1] ) {
7                 int t  = a[j];
8                 a[j]   = a[j+1];
9                 a[j+1] = t;
10             }
11    }
12
13    public static void main ( String [] args ) {
14       int [] array = new int [args.length];
15       for (int i = 0; i < args.length; i++)
16          array[i] = Integer.parseInt ( args [i] );
17
18       sort (array);
19
20       for (int k = 0; k < array.length; k++)
21           System.out.print(array [k] + "␣");
22       System.out.println();
23    }
24 }
```