

# Java Programming: Strings, Arrays, and others

Desenvolvimento de Software e Sistemas Móveis (DSSMV)

Licenciatura em Engenharia de Telecomunicações e Informática

LETI/ISEP

2025/26

Paulo Baltarejo Sousa and Carlos Filipe Freitas

`{pbs,caf}@isep.ipp.pt`



Instituto Superior de  
Engenharia do Porto

P.PORTO

## Disclaimer

### Material and Slides

Some of the material/slides are adapted from various:

- Presentations found on the internet;
- Books;
- Web sites;
- ...

# Outline

- 1 Strings
- 2 Arrays
- 3 Array Lists
- 4 Converting to ...
- 5 Random Numbers
- 6 Programming Issues
  - Primitive Data Types
  - Integer Division
- 7 Bibliography

# Strings

## What is a string?

- Strings are sequences of characters.
- `String` type.

```
String name = "Harry";
```

- `name` is a string variable.
  - `"Harry"` is a string literal.
- The number of characters in a string is called the **length** of the string.

```
int n = name.length();
```

- A string of length 0 is called the **empty** string.
    - It contains no characters and is written as `""`.

```
String name = "";
```

- String positions are counted starting with 0.

H	a	r	r	y
0	1	2	3	4

## Creating a String

- `String` is a Java class.
- There are two ways to create a `String` objects:
  - String literal

```
String str1 = "Welcome";
```

- String object is created by compiler.
- Using `new` keyword.

```
String str1 = new String("Welcome");
```

## Concatenation

- Use the + operator to concatenate strings; that is, to put them together to yield a longer string.

```
String fName = new String("Harry");  
String lName = new String("Morgan");  
String name = new String(fName + " " + lName);
```

- Whenever one of the arguments of the + operator is a string, the other argument is converted to a string.

```
String jobTitle = new String("Agent");  
int employeeId = 7;  
String bond = new String(jobTitle + employeeId);
```

## Reading a string

- Use the `next` method of the `Scanner` class to read a string containing a single word.

```
System.out.print("Please enter one word: ");  
String word = new String(in.next());
```

- Use the `nextLine` method of the `Scanner` class to read characters until it finds a new line character '`\n`'.

```
System.out.print("Please write a sentence: ");  
String sentence = new String(in.nextLine());
```



## Escape Sequences

- To include a quotation mark in a literal string, precede it with a backslash (\), like this:

```
String message = new String("He said \"Hello\"");
```

- The backslash is not included in the string. It indicates that the quotation mark that follows should be a part of the string and not mark the end of the string.
- The sequence \" is called an escape sequence.
- To include a backslash in a string, use the escape sequence \\, like this:

```
String path = new String("C:\\Temp\\Secret.txt");
```

## Strings and Characters

- Strings are sequences of Unicode characters.
- A character is a value of the type `char`.
  - Characters have numeric values
    - Character 'H' is number 72.
- Character literals are delimited by single quotes (' '), while strings are delimited by quotes (" ");
  - 'H' is a character, a value of type `char`.
  - "H" is a string containing a single character, a value of type `String`.
- The `charAt` method returns a `char` value from a string.

```
String name = new String("Harry");  
char start = name.charAt(0);  
char last = name.charAt(4);
```

- It sets `start` to the value 'H' and `last` to the value 'y'.

## Substrings

- Use the `substring` method to extract a part of a string.
  - The method call `str.substring(start, pastEnd)`
    - It returns a string that is made up of the characters in the string `str`, starting at position `start`, and containing all characters up to, but not including, the position `pastEnd`.

```
String greeting = new String("Hello, World!");  
String sub = greeting.substring(0, 5); // sub is "Hello"
```

H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

## Comparing strings

- `boolean equals(Object obj)`: Compares the string with the specified string and returns true if both matches else false.

```
String str1= new String("Hello");
String str2= new String("Hi");
if(str1.equals(str2)){
    //equals
}else{
    //different
}
```

- `int compareTo(String string)`: This method compares the two strings based on the Unicode value of each character in the strings.

```
String str1= new String("Hello");
String str2= new String("Hi");
String str3 = "Hello";
int var1 = str1.compareTo( str2 );
System.out.println(str1 + " & " + str2 + " comparison: "+var1);
int var2 = str1.compareTo( str3 );
System.out.println(str1 + " & " + str3 + " comparison: "+var2);
```

## Converting strings

- `int parseInt(String s)`: convert the string `s` to integer

```
String str3=new String("1234");  
int num3 = Integer.parseInt(str3);
```

- `String.valueOf(data type)`: This method returns a string representation of specified data type.

```
int ivar = 150;  
String str = new String(String.valueOf(ivar));
```

**Check:** TP2\_01.zip

# String Operations

Statement	Result	Comment
<code>string str = "Ja"; str = str + "va";</code>	str is set to "Java"	When applied to strings, + denotes concatenation.
<code>System.out.println("Please" + " enter your name: ");</code>	Prints Please enter your name:	Use concatenation to break up strings that don't fit into one line.
<code>team = 49 + "ers"</code>	team is set to "49ers"	Because "ers" is a string, 49 is converted to a string.
<code>String first = in.next(); String last = in.next(); (User input: Harry Morgan)</code>	first contains "Harry" last contains "Morgan"	The next method places the next word into the string variable.
<code>String greeting = "H &amp; S"; int n = greeting.length();</code>	n is set to 5	Each space counts as one character.
<code>String str = "Sally"; char ch = str.charAt(1);</code>	ch is set to 'a'	This is a char value, not a String. Note that the initial position is 0.
<code>String str = "Sally"; String str2 = str.substring(1, 4);</code>	str2 is set to "all"	Extracts the substring starting at position 1 and ending before position 4.
<code>String str = "Sally"; String str2 = str.substring(1);</code>	str2 is set to "ally"	If you omit the end position, all characters from the position until the end of the string are included.
<code>String str = "Sally"; String str2 = str.substring(1, 2);</code>	str2 is set to "a"	Extracts a String of length 1; contrast with <code>str.charAt(1)</code> .
<code>String last = str.substring( str.length() - 1);</code>	last is set to the string containing the last character in str	The last character has position <code>str.length() - 1</code> .

**Check:** <http://beginnersbook.com/2013/12/java-strings/>

# Arrays

## What is an Array?

- An array is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created.
  - After creation, its length is fixed.
- Each item in an array is called an element, and each element is accessed by its numerical index.
- The elements of arrays are numbered starting at 0.



- The expression `a.length` yields the length of the values array. Note that there are no parentheses following length.



## Declaring and Accessing (I)

**Syntax** To construct an array: `new typeName[length]`


To access an element: `arrayReference[index]`

Name of array variable  
 Type of array variable `double[]` values = new `double`[10];  
 Element type Length  
`double[]` moreValues = { 32, 54, 67.5, 29, 35 };  
 Use brackets to access an element.  
 values[i] = 0;  
 The index must be  $\geq 0$  and  $<$  the length of the array.  
 See page 314.

List of initial values

- An array index must be at least zero and less than the length of the array.
- A bounds error, which occurs if you supply an invalid array index, can cause your program to terminate.

## Declaring and Accessing (II)

<code>int[] numbers = new int[10];</code>	An array of ten integers. All elements are initialized with zero.
<code>final int LENGTH = 10; int[] numbers = new int[LENGTH];</code>	It is a good idea to use a named constant instead of a “magic number”.
<code>int length = in.nextInt(); double[] data = new double[length];</code>	The length need not be a constant.
<code>int[] squares = { 0, 1, 4, 9, 16 };</code>	An array of five integers, with initial values.
<code>String[] friends = { "Emily", "Bob", "Cindy" };</code>	An array of three strings.
 <code>double[] data = new int[10];</code>	<b>Error:</b> You cannot initialize a <code>double[]</code> variable with an array of type <code>int[]</code> .

```
int[] a = new int [12];
for (int i = 0; i<a.length; i++ )
    a[i] = i + 1;
for (int i = 0; i<a.length; i++ )
    System.out.println(a[i]);
}
```

# Array Lists

## What is an Array List?

- It is a class.
- It is a **container** that store lists of objects.
- It offers two significant advantages:
  - It can grow and shrink as needed.
  - The `ArrayList` class supplies methods for common tasks, such as inserting and removing elements.
- The `ArrayList` class is a generic class: `ArrayList<Type>` collects elements of the specified type.
  - The `Type` is a class.

```
ArrayList<String> names = new ArrayList<String>();  
names.add("Emily");  
names.add("Bob");  
names.add("Cindy");  
  
for (int i = 0; i < names.size(); i++) {  
    System.out.println(names.get(i));  
}
```

# Declaring

**Syntax** To construct an array list: `new ArrayList<typeName>()`

To access an element: `arraylistReference.get(index)`  
`arraylistReference.set(index, value)`

Variable type      Variable name      An array list object of size 0

`ArrayList<String> friends = new ArrayList<String>();`

Use the  
get and set methods  
to access an element.

```
friends.add("Cindy");
String name = friends.get(i);
friends.set(i, "Harry");
```

The add method  
appends an element to the array list,  
increasing its size.

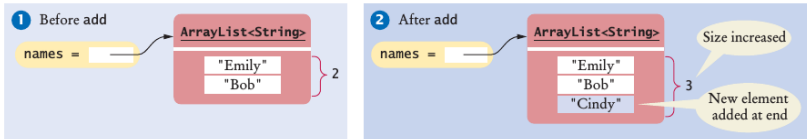
The index must be  $\geq 0$  and  $< \text{friends.size}()$ .

**Check:** <http://beginnersbook.com/2013/12/java-arraylist/>

## Operations (I)

```
ArrayList<String> names = new ArrayList<String>();  
names.add("Emily");  
names.add("Bob");
```

```
names.add("Cindy");  
names.set(2, "Carolyn");
```



## Operations (II)

```
names.add(1, "Ann");  
names.remove(1);
```

1 Before add

names =

ArrayList<String>

"Emily"  
"Bob"  
"Carolyn"

2 After names.add(1, "Ann")

names =

ArrayList<String>

"Emily"  
"Ann"  
"Bob"  
"Carolyn"

New element  
added at index 1

Moved from index 1 to 2

Moved from index 2 to 3

3 After names.remove(1)

names =

ArrayList<String>

"Emily"  
"Bob"  
"Carolyn"

Moved from index 2 to 1

Moved from index 3 to 2

## Working with Array Lists

<code>ArrayList&lt;String&gt; names = new ArrayList&lt;String&gt;();</code>	Constructs an empty array list that can hold strings.
<code>names.add("Ann");</code> <code>names.add("Cindy");</code>	Adds elements to the end of the array list.
<code>System.out.println(names);</code>	Prints [Ann, Cindy].
<code>names.add(1, "Bob");</code>	Inserts an element at index 1. names is now [Ann, Bob, Cindy].
<code>names.remove(0);</code>	Removes the element at index 0. names is now [Bob, Cindy].
<code>names.set(0, "Bill");</code>	Replaces an element with a different value. names is now [Bill, Cindy].
<code>String name = names.get(i);</code>	Gets an element.
<code>String last = names.get(names.size() - 1);</code>	Gets the last element.
<code>ArrayList&lt;Integer&gt; squares = new ArrayList&lt;Integer&gt;();</code> <code>for (int i = 0; i &lt; 10; i++)</code> <code>{</code> <code>squares.add(i * i);</code> <code>}</code>	Constructs an array list holding the first ten squares.



## Wrappers

- You **cannot directly insert primitive type values** such as `int`, `char`, or `boolean` values—into array lists.
  - For example, you cannot form an `ArrayList<double>`.
  - You must use one of the **wrapper classes**

Primitive Type	Wrapper Class
<code>byte</code>	<code>Byte</code>
<code>boolean</code>	<code>Boolean</code>
<code>char</code>	<code>Character</code>
<code>double</code>	<code>Double</code>
<code>float</code>	<code>Float</code>
<code>int</code>	<code>Integer</code>
<code>long</code>	<code>Long</code>
<code>short</code>	<code>Short</code>

- Note that the wrapper class names start with uppercase letters, and that two of them differ from the names of the corresponding primitive type: `Integer` and `Character`.
- To collect numbers in array lists, you must use wrapper classes.

## Auto-boxing

- Conversion between primitive types and the corresponding wrapper classes is automatic.
- This process is called **auto-boxing**.
  - For example, if you assign a `double` value to a `Double` variable, the number is automatically **put into a box**.

```
Double wrapper = 29.95;
```

- Conversely, wrapper values are automatically **unboxed** to primitive types.

```
double x = wrapper;
```

## Choosing Between Array Lists and Arrays

- Use an array
  - If the **size of a collection never changes**;
  - If you collect a **long sequence of primitive type values** and you are concerned about efficiency.
- Use an array list
  - Otherwise.
- Comparing

Operation	Arrays	Array Lists
Get an element.	<code>x = values[4];</code>	<code>x = values.get(4);</code>
Replace an element.	<code>values[4] = 35;</code>	<code>values.set(4, 35);</code>
Number of elements.	<code>values.length</code>	<code>values.size()</code>
Number of filled elements.	<code>currentSize</code> (companion variable, see Section 7.1.4)	<code>values.size()</code>
Remove an element.	See Section 7.3.6.	<code>values.remove(4);</code>
Add an element, growing the collection.	See Section 7.3.7.	<code>values.add(35);</code>
Initializing a collection.	<code>int[] values = { 1, 4, 9 };</code>	No initializer list syntax; call <code>add</code> three times.

**Converting to ...**

## An array

- From a String

```
String testString = new String("This Is Test");  
char[] stringToCharArray = testString.toCharArray();
```

- From an ArrayList

```
ArrayList<Integer> list = new ArrayList<Integer>();  
...  
int array[] = new int [list.size()];  
for(int i =0; i < list.size();i++){  
    array[i] = list.get(i);  
}
```

```
ArrayList<Integer> list = new ArrayList<Integer>();  
...  
Integer[] resultArray = new Integer[list.size()];  
resultArray = list.toArray(resultArray);
```

```
ArrayList<String> stringList = new ArrayList<String>();  
...  
String[] stringArray = new String[stringList.size()];  
stringArray = stringList.toArray(stringList);
```

# An Array List

- From an array of String

```
String[] array = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",  
    "Oct", "Nov", "Dec"};  
ArrayList<String> list = new ArrayList<String>(Arrays.asList(array));  
for(int i =0; i < list.size();i++){  
    System.out.println(list.get(i));  
}
```

- From an array of int

```
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
ArrayList<Integer> list = new ArrayList<Integer>();  
for(int i =0; i < arr.length;i++){  
    list.add(arr[i]);  
}  
for(int i =0; i < list.size();i++){  
    System.out.println(list.get(i));  
}
```

# Random Numbers

## Generate random numbers

- The `Random` class of the Java library implements a **random number generator** that produces numbers that appear to be completely random.
- To generate random numbers, you construct an object of the `Random` class, and then apply one of the following methods:
  - `nextInt(n)`: A random integer between the integers 0 (inclusive) and `n` (exclusive).
  - `nextDouble()`: A random floating-point number between 0 (inclusive) and 1 (exclusive).

```
Random randomGenerator = new Random();  
for (int i = 0; i < 10; i++){  
    int randomInt = 1 + randomGenerator.nextInt(100);  
}
```

Check: TP2\_02.zip



# Programming Issues

## Overflow

- A numeric computation overflows if the result falls outside the range for the number type.

```
int n = 1000000;  
System.out.println(n * n); // Prints -727379968, which is clearly wrong
```

- The product  $n * n$  is  $10^{12}$ , which is larger than the largest integer (about  $2 * 10^9$ ).
- The result is truncated to fit into an `int`.
- There is no warning when an integer overflow occurs.

## Floating-point representation

- Rounding errors occur when an exact representation of a floating-point number is not possible.

```
double f = 4.35;  
System.out.println(100 * f); // Prints 434.99999999999994
```

- The problem arises because computers represent numbers in the binary number system.
  - In the binary number system, there is no exact representation of the fraction  $1/10$ , just as there is no exact representation of the fraction  $1/3 = 0.33333$  in the decimal number system.
- For this reason, the `double` type is not appropriate for financial calculations.

## Unintended Integer Division

- / is used for both integer and floating-point division.
- It is a common error to use integer division by accident.

```
int score1 = 10;
int score2 = 4;
int score3 = 9;
double average = (score1 + score2 + score3) / 3; // Error
System.out.println("Average score: " + average); // Prints 7.0, not
7.666666666666667
```

- Solution:

```
double total = score1 + score2 + score3;
double average = total / 3;
```

or

```
double average = (score1 + score2 + score3) / 3.0;
```

# Bibliography

## Resources

- "Big Java: Early Objects", 6th Edition by Cay S. Horstmann
- "Java™:The Complete Reference", 7th Edition,Herbert Schildt
- "Java™Programming", 7th Edition, Joyce Farrell
- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>
- <http://beginnersbook.com/java-tutorial-for-beginners-with-examples/>
- <https://www.lepoint.net/index.html>
- <https://junit.org/junit5/docs/current/user-guide/>