UNIVERSITÄT BERN

Java

Arrays and Lists

Lecturer: **Nataliia Stulova**
Teaching assistant: **Mohammadreza Hazirprasand**

Software Composition Group
University of Bern
23 September 2020

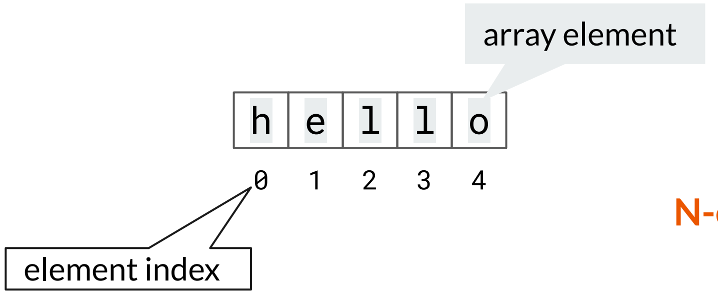
MASTER IN COMPUTER SCIENCE
OF THE UNIVERSITIES OF BERN, NEUCHÂTEL AND FRIBOURG

Software Skills Lab

Java Arrays

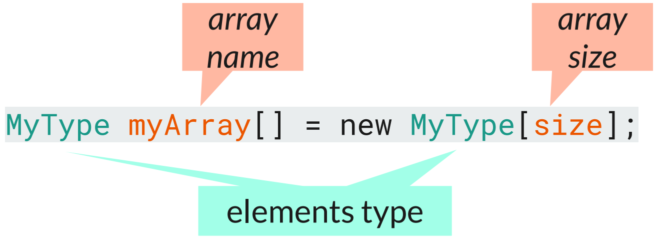
Array data structure

- a data structure to store a fixed number of elements of the same type
- elements are accessed by their relative position (***random access***) - each element is independent of others



N-elements array indices range from `0` to `N-1`

Java arrays



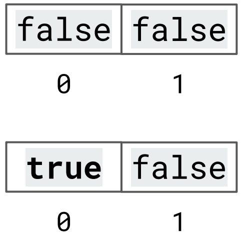
```
MyType myArray[] = new MyType[size];
```

```
MyType myArray[];
myArray = new MyType[size];
```

On creation arrays of *primitive* types are filled with ***default values***:

```
boolean status[];
status = new boolean[2];
```

```
status[0] = true;
```



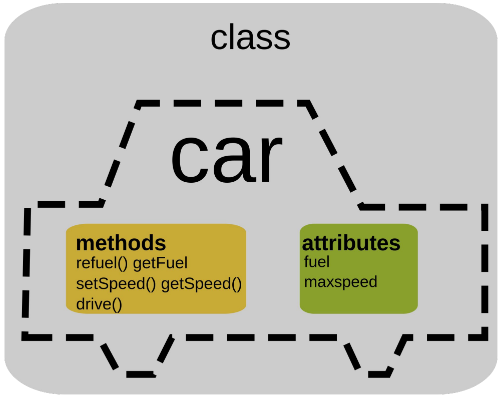
Creating Java arrays

Arrays of primitive types

```
int nums[] = new int[2];

nums[0] = 23;
nums[1] = 9;

int nums[] = {23, 9};
```

Arrays of objects

```
Car parking[] = new Car[20];

parking[0] = new Car();
parking[0].setSpeed(0);

Car truck = new Car();
truck.fuel = 20;
parking[1] = truck;
```



Multi-dimensional arrays

Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other array

```
MyType matrix[]..[] = new MyType[s1]..[sN];
```

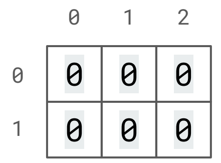
number of dimensions

each dimension sizes

Examples: spreadsheets, games (like sudoku), timetables, images

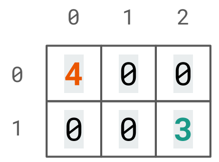
rows *columns*

```
int matrix[][] = new int[2][3];
```



	0	1	2
0	0	0	0
1	0	0	0

```
matrix[0][0] = 4;
matrix[1][2] = 3;
```



	0	1	2
0	4	0	0
1	0	0	3

java.util.Arrays (Java SE 9)

Reference Javadoc: https://docs.oracle.com/javase/9/docs/api/java/util/Arrays.html

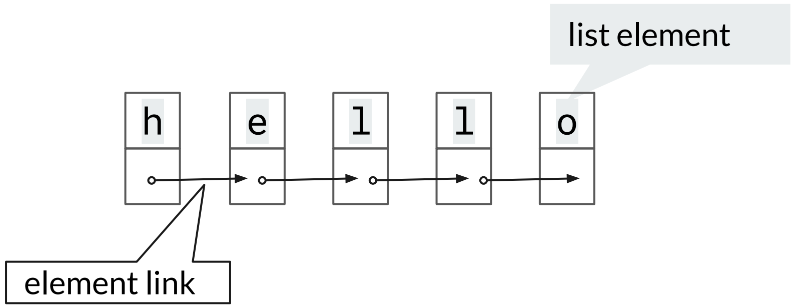
A library class that provides various useful operations on arrays:

- `fill()`
- `sort()`
- `binarySearch()`
- `copyOf()`
- `equals()`

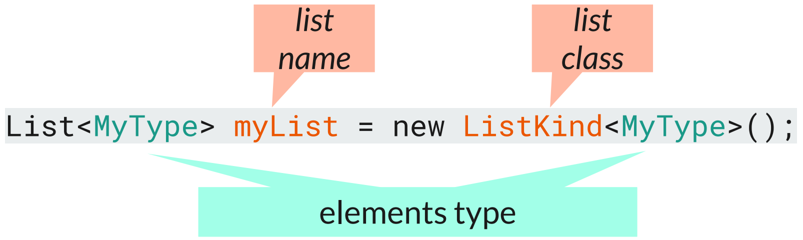
Java Lists

Linked list data structure

- a data structure to store a *non*-fixed number of elements of the same type
- elements are accessed in their order (***sequential access***) - each element needs to be connected to the previous



Creating Java lists



```java
List<MyType> myList = new ListKind<MyType>();
```

```java
List<MyType> myList;
myList = new ListKind<MyType>();
```

- `List<...>` is an ***Interface*** - a blueprint of a class, that does not hold any implementation details
- `ListKind<...>` is a **Class** - a blueprint of an object, has attributes and methods, does not hold any values
- `myList` is an **Object** - an instance of the `ListKind<...>` class, holds concrete values in its attributes

Java lists: Classes VS Interfaces

```
interface List<E> {
   add(...);
   remove(...);
   ...
}
```

```
class LinkedList<E> {
  attributes, if any

   add(...) {...}
   remove(...) {...}
   ...
}
```

```
List<String> myList = new LinkedList<String>();
myList.add(“Potatoes”);
```

- `List<E>` is an ***Interface*** - a blueprint of a class, that does not hold any implementation details
- `LinkedList<E>` is a **Class** - a blueprint of an object, has attributes and methods, does not hold any values
- `myList` is an **Object** - an instance of the `LinkedList<E>` class, holds concrete values in its attributes

java.util.List

Reference Javadoc: https://docs.oracle.com/javase/9/docs/api/java/util/List.html

A library interface that provides various useful operations on lists:

- **`get()`**
- **`add()`**`, addAll()`
- **`remove()`**
- `contains(), containsAll()`
- `clone()`
- `equals()`

Accessing list elements

```
List<String> groceries = Arrays.asList("Potatoes", "Ketchup", "Eggs");
```

Loops

```
for (int i = 0; i < groceries.size(); i++) {
        System.out.println(groceries.get(i));
    }

for (String product : groceries) {
        System.out.println(product);
    }
```

Iterators

An interface to go through elements in a collection data structure:

- `hasNext()` method checks if there are any elements remaining in the list
- `next()` method returns the next element in the iteration

```
Iterator<String> groceriesIterator = groceries.iterator();

while(groceriesIterator.hasNext()) {
        System.out.println(groceriesIterator.next());
}
```

Summary and practice

What you should remember

Use arrays when:

- you know the number of elements…
- … or the number of elements will increase rarely
- you need fast access to individual elements

Use lists when:

- you do not know the number of elements
- you do not need fast access to individual elements

NEW `this` keyword: clarify the context

```
result[i][j] = this.matrix[i][j] + other.matrix[i][j]
```

Exercise 1: Arrays

Matrix multiplication

- write a class representing a 2D matrix
- attributes:
    - `int matrix[][]`
- methods:
    - `Matrix(int rows, int cols)` - constructor
    - `Matrix add(Matrix other)` - addition
    - `Matrix product(Matrix other)` - multiplication

https://en.wikipedia.org/wiki/Matrix_(mathematics)#Basic_operations

I/O

-

Tests (JUnit, class MatrixTest)

- dimensions mismatch
- 3 correct cases: 1-column matrix, 1-row matrix, a 2x3 matrix

NEW `static` keyword: helper methods (and no objects!)

`Double arMean = Averages.artihMean(ArrayList<E> nums)`

NEW boxed types: Integer, Float, Double....

Exercise 2: Lists

Computing various average values

- write a class `Averages` to compute various means: arithmetic, geometric, and harmonic
  https://en.wikipedia.org/wiki/Average
- methods:
  - `static Double arithMean(ArrayList<E> nums)`
  - `static Double geomMean(ArrayList<E> nums)`
  - `static Double harmMean(ArrayList<E> nums)`

I/O

- Read a sequence of numbers from `System.in`
- Print average values to `System.out`

Tests

-