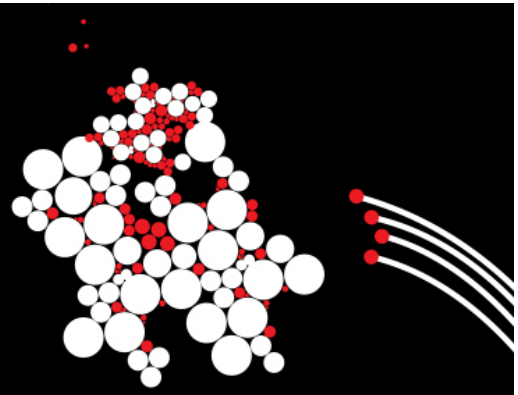
UNIVERSITY OF TWENTE.

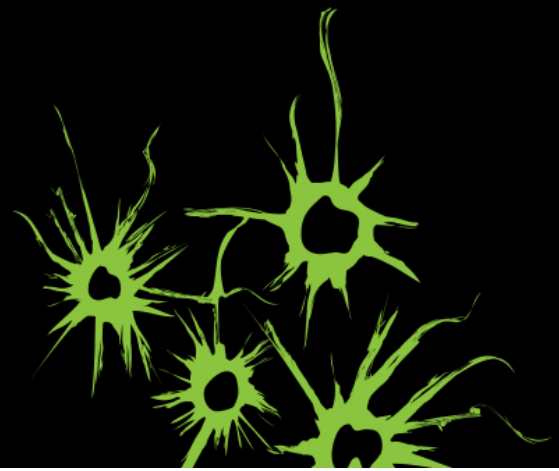


P4.1: ARRAYS AND LISTS

LUÍS FERREIRA PIRES

201700117-1B MODULE 2: SOFTWARE SYSTEMS

3 DECEMBER 2019



PROGRAMMING LINE OVERVIEW

Week 1 Values and variables Control flow	**Week 2** Classes and objects Testing	**Week 3** Interfaces and Inheritance Subtyping Security 1
Week 4 Arrays and Lists List implementations Collections	**Week 5** Stream I/O and MVC Exceptions Security 2	**Week 6** Concurrency Project kick-off IDE Tips & Tricks
Week 7 Basic Networking Networking and Multithreading GUIs	**Week 8/9** Advanced Java facilities Test	**Week 10** Project Test resit

LIST CONCEPT
INTUITIVELY

- List is a placeholder for (references to) things
- Things in a list have the same type
- Things in a list have a position
- Position of a thing in list can change

	Club	MP	W	D	L	GF	GA	GD	Pts	Last 5
1	Ajax	15	13	2	0	52	12	40	41	W W W W W
2	AZ Alkmaar	15	11	2	2	35	8	27	35	W W W W W
3	PSV Eindhoven	15	8	4	3	32	19	13	28	D W L D L
4	Willem II	15	8	2	5	23	21	2	26	W D W L W
5	Heracles	15	7	3	5	29	22	7	24	W L W L W
6	Heerenveen	15	6	6	3	24	18	6	24	W L W D D
7	Feyenoord	15	6	6	3	27	25	2	24	W D W W L
8	Utrecht	15	7	2	6	28	22	6	23	L L L W W
9	Vitesse	15	7	2	6	26	24	2	23	L L L L L
10	Groningen	15	6	3	6	18	16	2	21	L D W W D
11	Sparta	15	5	4	6	24	28	-4	19	L W L D L
12	Twente	15	5	3	7	26	31	-5	18	L L W L W
13	FC Emmen	15	4	3	8	18	30	-12	15	D D L W L
14	Fortuna Sittard	15	4	3	8	20	35	-15	15	W L W L W
15	Zwolle	15	4	1	10	22	34	-12	13	L W L L L
16	VVV	15	4	0	11	14	39	-25	12	L W L L L
17	Den Haag	15	3	2	10	17	30	-13	11	L D L D W
18	RKC Waalwijk	15	2	2	11	16	37	-21	8	W D L W L

LISTS IN PROGRAMMING LANGUAGES

- Lists are finite
  - Have a length (= number of elements), which can be 0 (“empty list”)
- Lists define a sequence
  - Elements in a list are in a certain order
  - Elements are numbered (indexed)
  - In Java, numbering starts with 0 (‘offset’)
- Elements in a list are homogenous
  - From the same (reference) type (directly or indirectly)

Easier to find elements in memory!

ARRAYS

- In Module 1 (Algorithms pearl) you learned to use arrays in Python
  - An array is a data structure to store elements of the same type
  - Array elements are stored and accessed in sequence
  - Arrays can be used to implement lists
  - Examples: array of `int`, `float`, `char`, `String`, `Account`, `Room`
- Java arrays have the same purpose and are similar to Python arrays (not exactly the same!)
  - Have to be declared and created before being used

ARRAY
DECLARATION AND CREATION

Declaration

- Defines an identifier to refer to an array

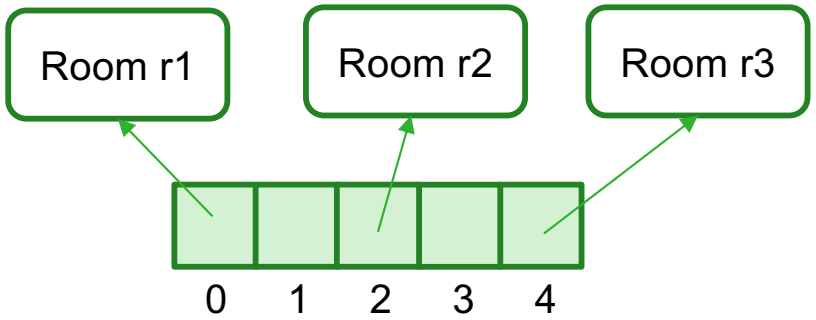
```
int[] values; String[] args; Room[] rooms;
```

Creation

- Allocates memory positions for the elements of the array

```
int[] values = new int[50];
String[] args = new String[10];

Room[] rooms = new Room[5];
```



ARRAY
USAGE

- Read values from an array

```
int i = values[3]; // copy 4th value of values to i
Room r = rooms[2]; // copy 3th reference of rooms to r
```

- Modify values from an array

```
values[3] = i;  // copy i to 4th position of values
rooms[2] = r;   // copy reference r to 3th position of rooms
```

ARRAY ITERATION

- 'Classic' pattern: iterate over the elements of an array

```
for (int i = 0; i < values.length; i++) {
     values[i] = i + 1;
}
```

- Array initialisation: array elements have to be created to be used

```
for (int i = 0; i < rooms.length; i++) {
     rooms[i] = new Room(101+i);
}
```

ARRAYS AS METHOD PARAMETERS

EXAMPLE

```
public static int sumValues(int[] values){
  int sum = 0;
  for(int i = 0; i < values.length; i++){
      sum += values[i];
  }
  return sum;
}
```

Array is passed as parameter

Array values are used

ARRAY LIMITATIONS

WHAT DO YOU HAVE TO DO IF YOU WANT TO…

- Remove an element of an arbitrary position of an array without creating an empty space?
  - Shift all elements to the beginning of the array one-by-one!
- Add an extra element to the array (list) when all positions are already taken?
  - Create new (bigger array), copy all elements and add new element
- You have to do all the manipulations by hand!!!

ALTERNATIVE: IMPLEMENTATION OF `List` INTERFACE

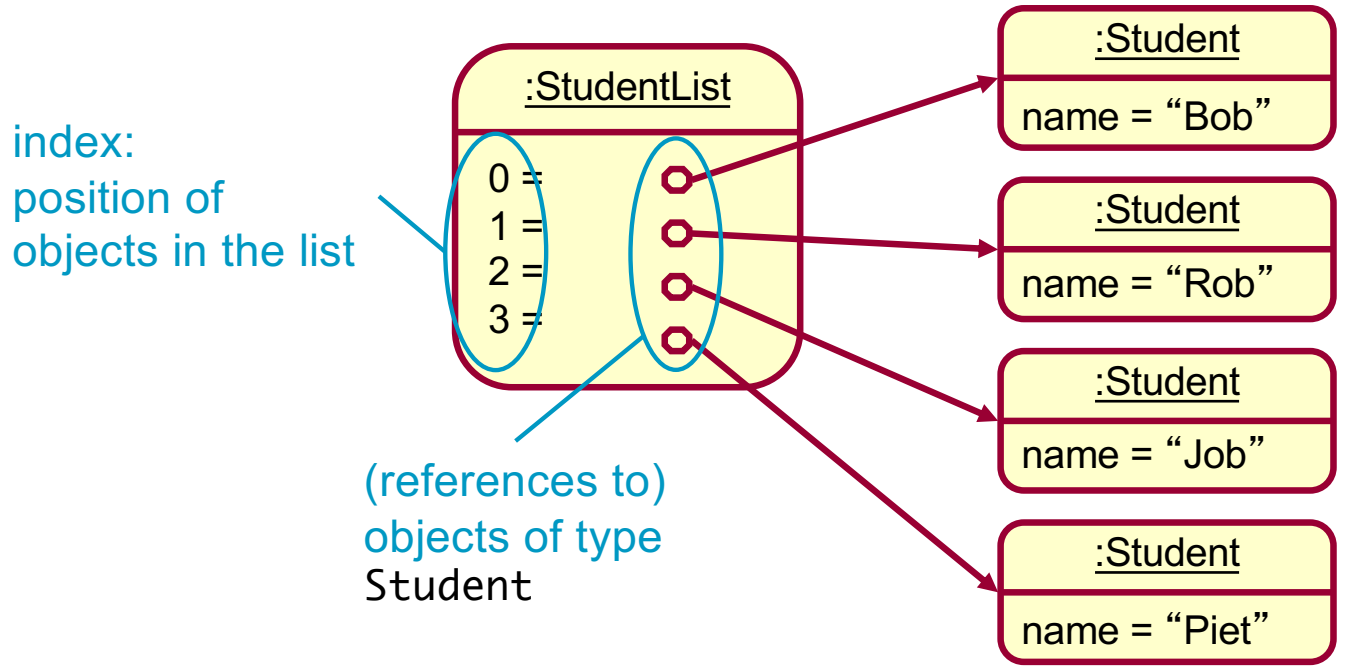
Principle

- Separate the abstract concept of a list and its possible implementations

- Now we discuss the abstract list concept and the `List` interface, in the next lecture we discuss techniques to implement (dynamic) lists

EXAMPLE

SUPPOSE WE WANT A LIST OF STUDENTS



LIST PROPERTIES AND METHODS

- A list is a (complex) data structure → instance of a reference type
  - What is the class of a list?
  - What are the properties of a list?
  - What are the queries?
  - What are the commands?
- Java Collections Framework → interfaces, implementations and algorithms to implement collections (sets, bags, lists, etc.)
- List is an implementation of the `List` interface

LIST METHODS (FOR A LIST OF STUDENTS)

SUPPOSE WE IMPLEMENT THE NECESSARY QUERIES AND COMMANDS

Queries (amongst others)

- Get the length of the list
- Test if the list is empty
- Get `Student` in position `i` (index `i`)

Commands (amongst others)

- Add `Student` to the end of the list
- Replace (set) `Student` in position `i`

GENERICS
MOTIVATION

- Fine that we have a list of `Student`s now, but we want to reuse the list methods to make lists of elements of different types (`Room`, `Date`, etc.)
- Generics have been introduced in Java 5 to allow an interface or class to have an element type as a 'parameter', to be replaced by an 'argument type' when the object of the class is created
- By defining the `List<E>` interface (`E` is a 'type parameter') we can use the same list implementations for lists of objects of different types
  - `List<Student>`, `List<Room>`, `List<Date>`, etc.

`List<E>` QUERIES

PACKAGE `java.utils`

- Number of elements: `public int size()`
  - Postcondition: `@ensures result >= 0`
- Is list empty? `public boolean isEmpty()`
  - Postcondition: `@ensures result == (this.size() == 0)`
- Get element at index `i`: `public E get(int i)`
  - Precondition: `@requires 0 <= i && i < this.size()`

List size cannot be negative

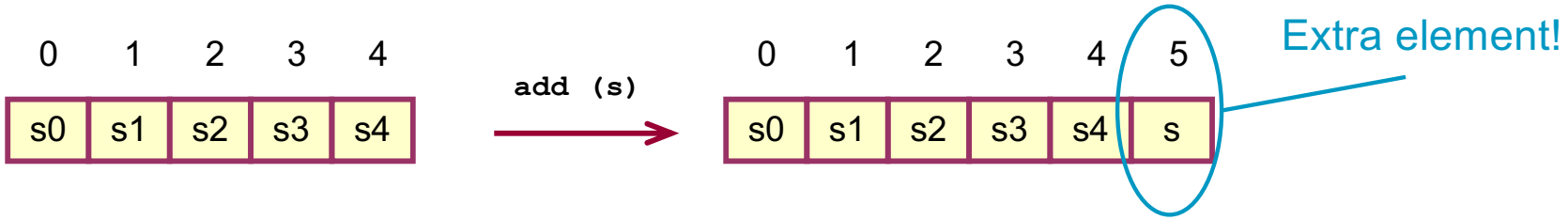
Index *i* should be inside the list!

List<E> COMMANDS

- Append element: `public void add(Student s)`
  - Postcondition

List grows one position, new element *s* is added to the end of the list and all other elements remain the same

```
@ensures this.size() == old.size() + 1 &
this.get(this.size() - 1).equals(s) &
(forall(int i; 0 <= i && i < old.size(); this.get(i).equals(old.get(i))
```



`List<E>` COMMANDS

- Set element at index `i`: `public void set(int i, E s)`
  - Precondition:

```
@requires 0 <= i && i < this.size()
```

  - Postcondition:

```
@ensures this.size() == old.size() &
this.get(i).equals(s) &
(forall int j; 0 <= j & j < this.size() & j != i;
    this.get(j).equals(\old.get(j))
```

List size does not change, new element *s* is placed in position *i* and all other elements remain the same

OTHER METHODS

- Index of first occurrence of element: `public int indexOf(E s)`
  - returns -1 if this list does not contain the element
- Does the list contain an element? `public boolean contains(E s)`
- Removes first occurrence of element: `public boolean remove(E s)`

- And many more (see Java 11 API)

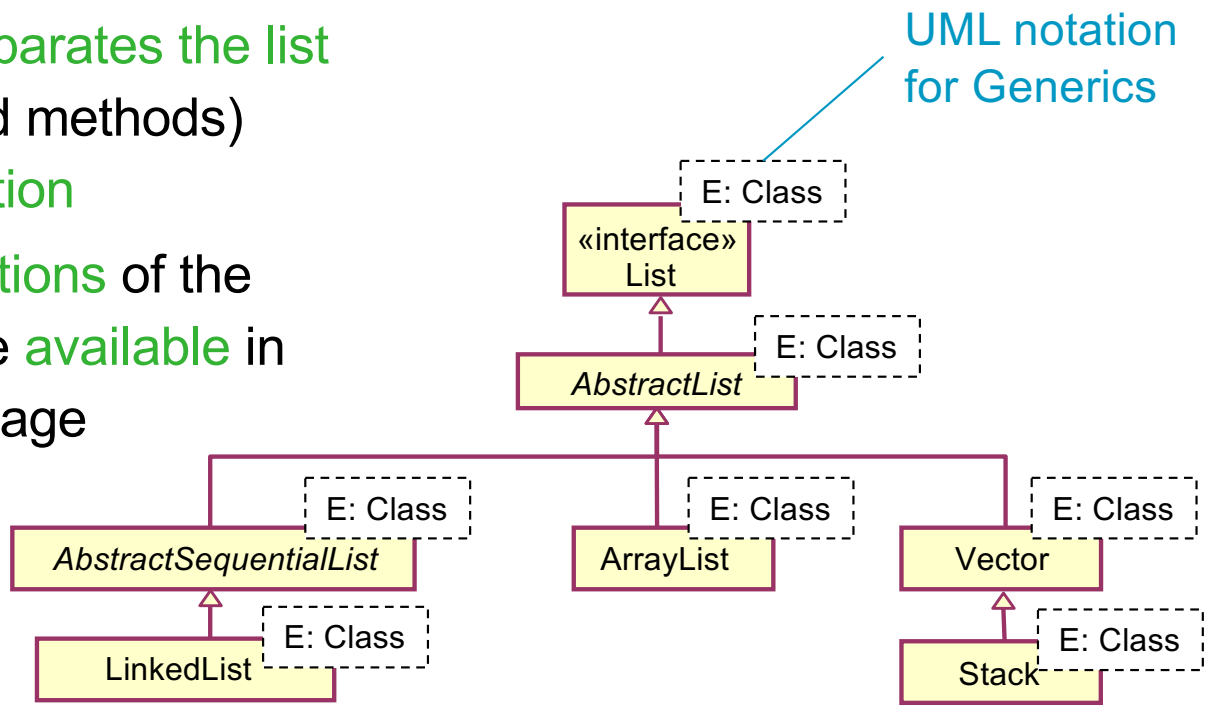
LIST OF PRIMITIVE VALUES

- Generics classes and interfaces(e.g., `List<E>` interface) only support reference types, not primitive types like `int`, `float`, `boolean`, `char`, etc.
- Java offers wrapper (reference) types for primitive types

  `int` → `Integer`
  `double` → `Double`
  `char` → `Character`
- List of integer values is denoted as `List<Integer>`
- List of double values is denoted as `List<Double>`, etc.

`List<E>` IMPLEMENTATIONS

- `List<E>` interface separates the list concept (and related methods) from its implementation
- Various implementations of the `List<E>` interface are available in the `java.utils` package



`List<E>` USAGE

- Interfaces cannot be used to instantiate objects, so we need classes
- Select a list implementation that fits your application!

```
List<Student> sl1 = new ArrayList<Student>(); // ok in most cases!
List<Student> sl2 = new LinkedList<Student>();
```

- By defining your reference as interface `List<E>` you can change the implementation later if necessary
- And don't forget to import the classes `java.utils.List`, `java.utils.ArrayList`, etc.

List<E> USAGE

FOR COMPLETENESS

```
List<Integer> values = new ArrayList<Integer>();
List<Room> rooms = new ArrayList<Room>();
```

- Read values from a list

```
int i = values.get(3); // copy 4th value of values to i
Room r = rooms.get(2); // copy 3th reference of rooms to r
```

- Modify values from a list

```
values.set(3, i);  // copy i to 4th position of values
rooms.set(2, r);   // copy reference r to 3th position of rooms
```

ITERATION OVER LISTS

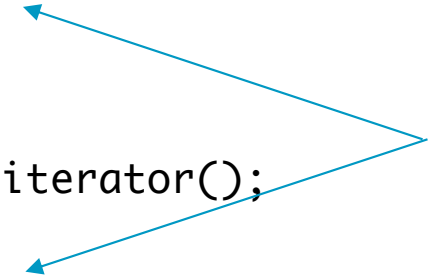
SHORTCUTS

- Java offers special syntax (shortcuts) to facilitate iteration over lists

```
java.util.List<Student> list;
for (Student s : list) {
  System.out.println(s);
}
```

- Alternative: `Iterator` Class

```
Iterator<Student> it = list.iterator();
while (it.hasNext()) {
  Student s = it.next();
  System.out.println(s);
}
```



Warning: Avoid changing list size while iterating over it!
For example, avoid calling `list.add(s);`

TAKE HOME MESSAGES

- Arrays in Java are similar to Python arrays, but different to use
- Arrays are handy only when memory allocation does not change (static lists) and cumbersome otherwise (dynamic lists)
- `List<E>` interface defines the methods that can be called on generic lists (independent of a specific type)
- To be used, `List<E>` must be instantiated with a type for `E` and refer to a list implementation (`ArrayList<E>`, `LinkedList<E>`, `Vector<E>`, etc.)