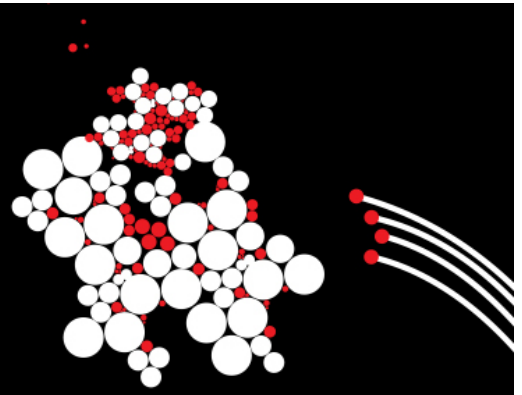
UNIVERSITY OF TWENTE.

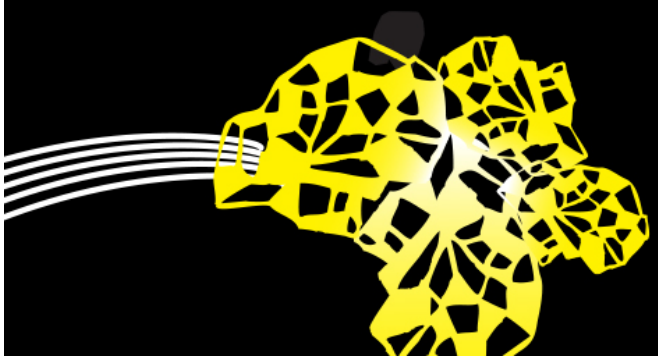


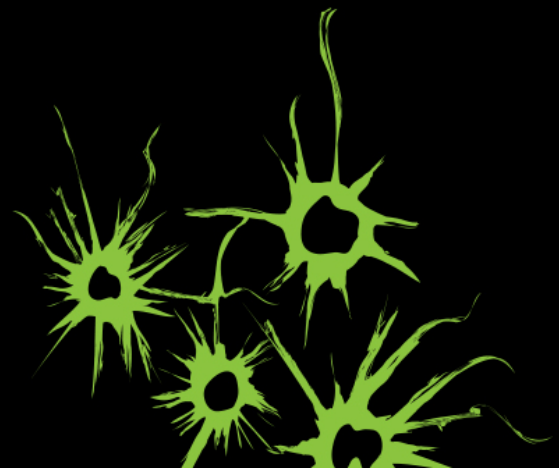
P4.2: LIST IMPLEMENTATIONS

LUÍS FERREIRA PIRES

201700117-1B MODULE 2: SOFTWARE SYSTEMS

3 DECEMBER 2019



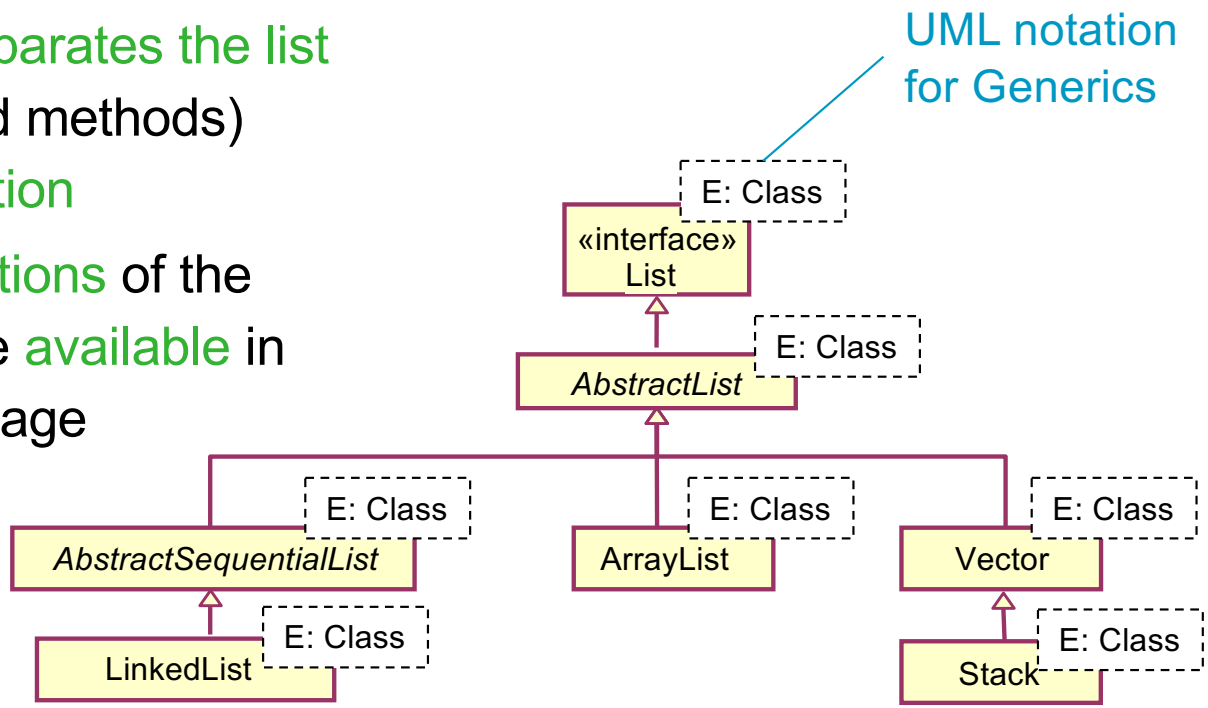


PROGRAMMING LINE OVERVIEW

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<E>` IMPLEMENTATIONS

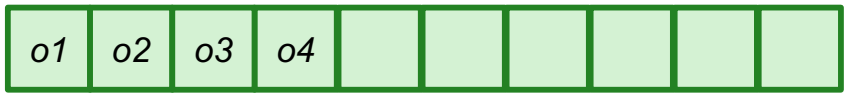
- `List<E>` interface separates the list concept (and related methods) from its implementation
- Various implementations of the `List<E>` interface are available in the `java.utils` package



List<E> IMPLEMENTATION WITH ARRAY

BASED ON ARRAYLIST<E>

```
public class SimpleArrayList<E> implements List<E> {
    public static final int INITIAL_SIZE = 10;
    private Object[] list;
    private int size;

    public SimpleArrayList() {
        this.list = new Object[INITIAL_SIZE];
        this.size = 0;
    }
```



list.length == *INITIAL_SIZE*

size == 4

List<E> IMPLEMENTATION WITH ARRAY

QUERIES

```
public int size() {
    return this.size;
}

public boolean isEmpty() {
    return size == 0;
}

public boolean contains(Object o) {
    boolean result = false;
    for (int i = 0; i < this.size && !result; i++) {
        if (list[i].equals(o))
            result = true;
    }
    return result;
}
```

```
public E get(int index) {
    if (index < 0 || index >= size)
        throw new ArrayIndexOutOfBoundsException();
    return (E) this.list[index];
}
```

`List<E>` IMPLEMENTATION WITH ARRAY

COMMANDS

```
public boolean add(E e) {
    if (this.size == this.list.length) {
        Object[] newList = new Object[this.list.length * 2];
        for (int i = 0; i < size; i++)
            newList[i] = this.list[i];
        list = newList;
    }
    this.list[this.size++] = e;
    return true;
}
```

allocate twice the space (`ArrayList` uses a more complex algorithm)

copy all elements to new array (could be done with `System.arraycopy()`)

COPYING LISTS

ALIAS

```
List<Room> rl1 = new ArrayList<Room>();
… // Insert elements in the list
List <Room> rl2 = rl1;
```

- Consequence
  - Changes in `rl2` are reflected in `rl1`
- Example
  - Command `rl2.set(2,r1)` also changes the third element of `rl1`!

COPYING LISTS

CLONING (SHALLOW COPY)

```
List<Room> rl1 = new ArrayList<Room>();
... // Insert elements in the list
List<Room> rl3 = (List<Room>) ((ArrayList<Room>) rl1).clone();
```

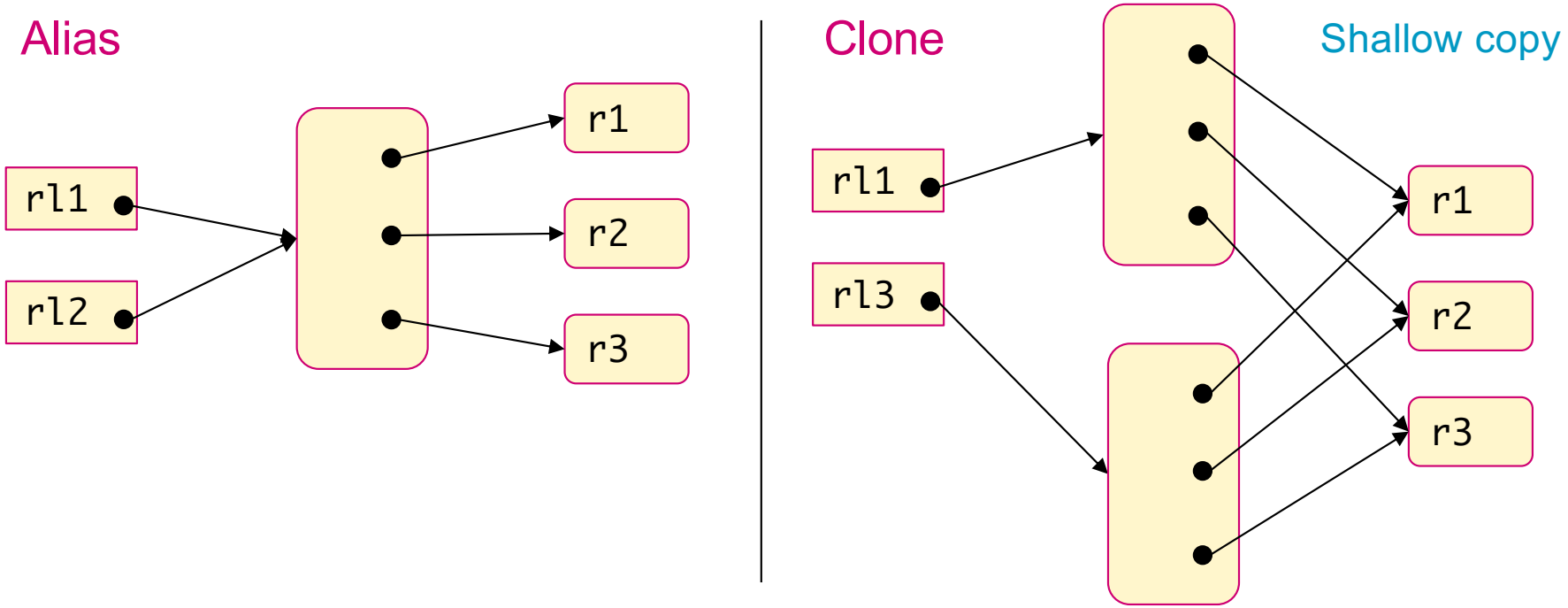
Necessary: `clone` needs a `Cloneable` object

Necessary: `clone` returns `Object`

- Consequence
  - Changes in `rl3` are not reflected in `rl1`
- Example
  - Command `rl3.set(2,r1)` does not change the third element of `rl1`!

COPYING LISTS

ALIAS VERSUS CLONING



COPYING LISTS

CLONING EXAMPLE

```
public Object clone() {
    SimpleArrayList<E> v = new SimpleArrayList<E>();
    for (int i = 0; i < this.size; i++)
        v.add((E) this.list[i]);
    return v;
}
```

cloneable object!

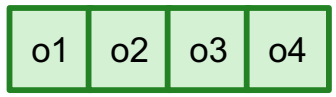
create new list

copy all elements to new list

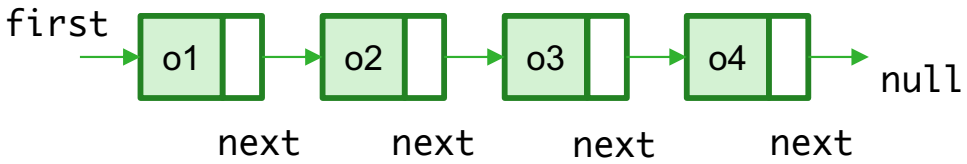
LINKED LIST

ALTERNATIVE STRUCTURE TO STORE LIST ELEMENTS

- List implementation stores information in a certain order
- Alternatively, we can store the information in contiguous areas (array) or in nodes, so that each node points to where the next piece of information is stored



Array list



Linked list

SIMPLE LINKED LIST

BASED ON NIÑO & HOSCH

```
public class SimpleLinkedList<E> implements List<E>{

    private class Node {
        E element;
        Node next;

        public Node (E element) {
            this.element = element;
            this.next = null;
        }
    }

    private int size;
    private Node first;

    public SimpleLinkedList() {
        this.size = 0;
        this.first = null;
    }
```

nested `Node` class only used by `SimpleLinkedList` (`private`)

properties: `size` and `first` Node

SIMPLE LINKED LIST

QUERIES

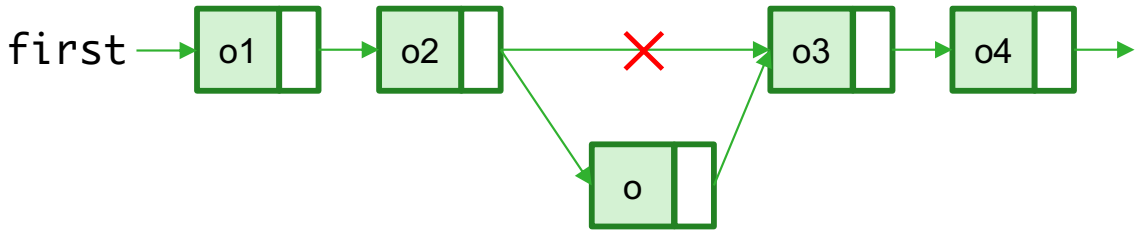
private help method

```
private Node getNode (int pos) {
    Node p = first;
    for (int i = 0; i != pos; i++ )
        p = p.next;
    return p;
}

public int size() {
    return this.size;
}

public boolean isEmpty() {
    return this.size == 0;
}

public boolean contains(Object o) {
    boolean result = false;
    for (Node p = first; p != null && !result; p=p.next) {
        if (p.element.equals(o))
            result = true;
    }
    return result;
}
```

SIMPLE LINKED LIST

ADD COMMAND

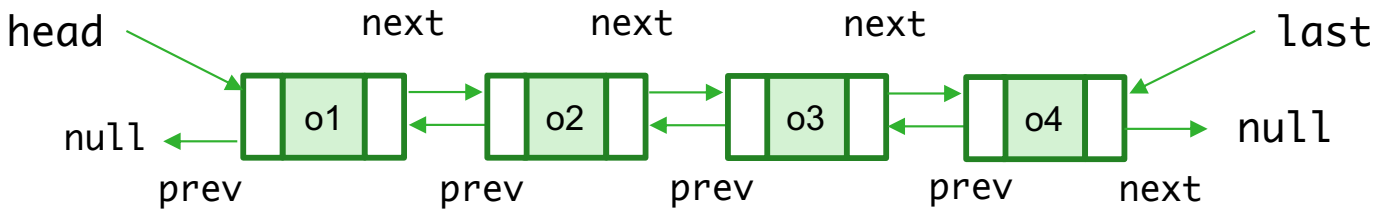
```
public void add(int index, E element) {
    Node newNode = new Node(element);
    if (index == 0) {
        newNode.next = first;
        first = newNode;
    } else {
        Node p = getNode(index - 1);
        newNode.next = p.next;
        p.next = newNode;
    }
    this.size++;
}
```



DOUBLE LINKED LIST
MOTIVATION

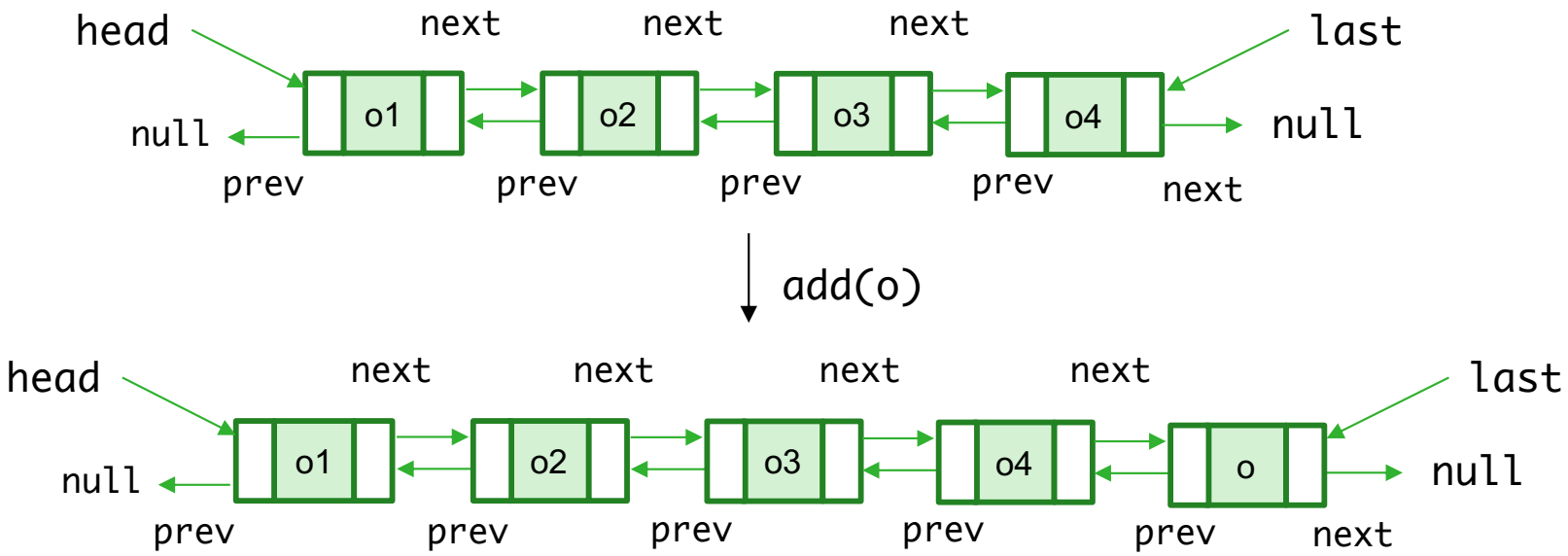
- In a linked list adding an element to the end of the list implies traversing the whole list → How can this be improved?
- To speed this up, double linked lists can be defined
- Node with pointers to both previous and next nodes

`LinkedList<E>` is defined like this!



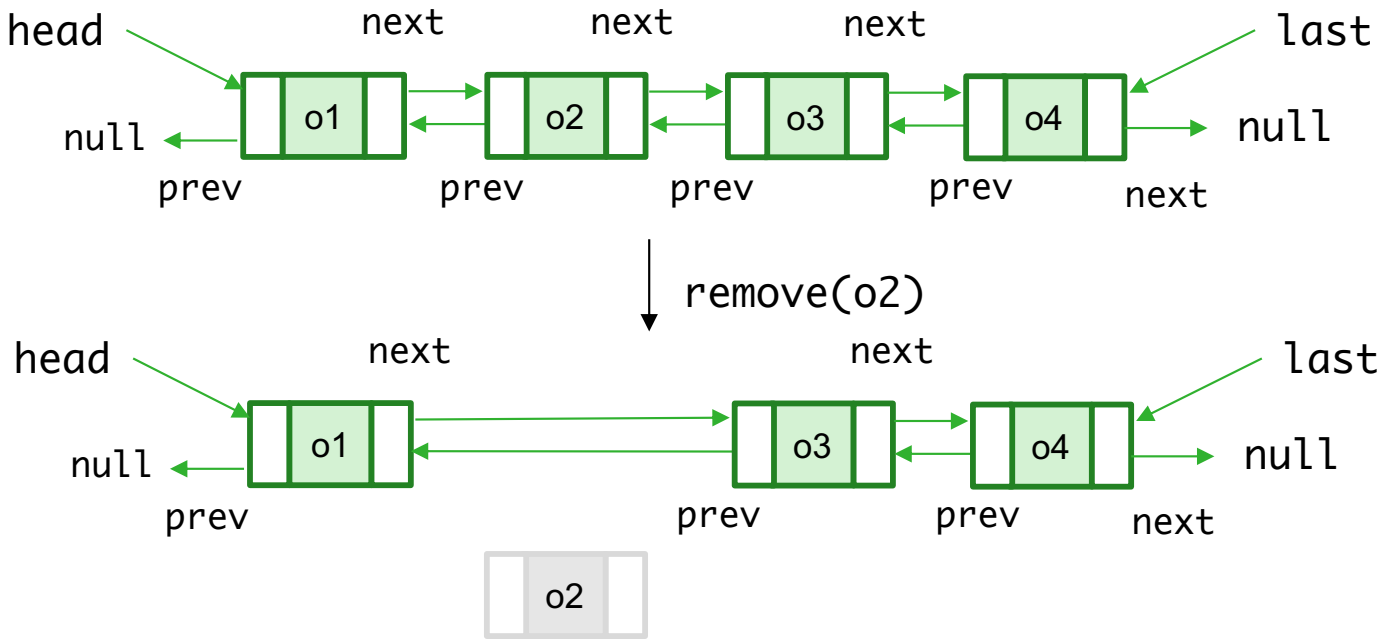
DOUBLE LINKED LIST

ADD ELEMENT (TO THE END)



DOUBLE LINKED LIST

REMOVE ELEMENT



ARRAY (DOUBLE) LINKED LISTS
COMPARISON

- In arrays it is easy to find an element, but difficult to insert or remove elements → a lot of copying!

- In (double) linked lists it is easy to insert or remove elements, but it is difficult to find elements → visit a lot of nodes before finding!

TAKE HOME MESSAGES

- Many different implementations of the `List<E>` interface are possible
- Implementations based on arrays, such as the `ArrayList<E>`, are more suitable when the elements are not often added and removed, and a lot of searching is done
- Implementations based on linked lists, such as the `LinkedList<E>`, are more suitable when elements are often added and removed, but searching is limited
- In practice, for your programs the differences are negligible!