Recap: Iterator and Iterable

- Before: Iterating via `for`/`while` loop
  - 'Looping logic' is in the code that executes the loop
- Iterator: An interface that represents an iteration
  - Easy to be used in conjunction with `while` loops
  - Two (central) methods: `boolean hasNext()` and `T next()`
  - Allows encapsulating looping conditions in an object
- Iterable: Signifies that one can iterate over an instance of the class
  - `Iterator<T> iterator()` returns an iterator

```
for (Student s : Course) {

}
```

Session 5: Generics and Collections, part 1

Fortgeschrittene Programmierung (Java 2)

Nils Reiter
`nils.reiter@uni-koeln.de`

May 10, 2023

INSTITUT FÜR DIGITAL HUMANITIES UNIVERSITÄT ZU KÖLN

CRETA CENTER FOR REFLECTED TEXT ANALYTICS

Section 1

Generics

Generics

Yes, this was mentioned in Winter,
but there wasn't an exercise about it

Generics

Yes, this was mentioned in Winter, but there wasn't an exercise about it

Motivation

- Duplicating code is bad
  - Errors fixed in one copy are not fixed in the other
  - Disk space
- Many things we do are similar, but for different types
  - E.g., collecting things, iterating over them, ...

Generics

Yes, this was mentioned in Winter, but there wasn't an exercise about it

Motivation

- Duplicating code is bad
  - Errors fixed in one copy are not fixed in the other
  - Disk space
- Many things we do are similar, but for different types
  - E.g., collecting things, iterating over them, ...

Generics

- Method to write 'template classes'
- Instantiated for different types
- Syntax: `Iterator<T>`, `MyClass<E extends Car>`, ...
  - `T`, `E` are variable names for class names
  - Only known at compile time
    - I.e., while we implement a generic class, we don't know what type it is used for

demo

Java Collections Framework

A collection is an object that represents a group of objects (such as the classic Vector class). A collections framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details.

Javadoc

Java Collections Framework

A collection is an object that represents a group of objects (such as the classic Vector class). A collections framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details.

Javadoc

Benefits

- Reduces programming effort
- Increases performance
- Fosters software reuse

Interfaces

`java.util.Collection`

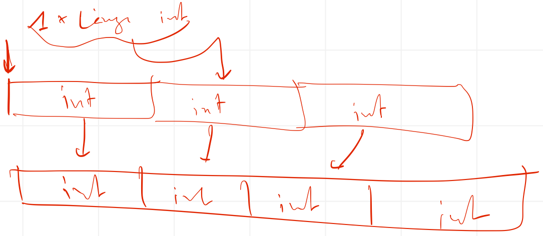
- `java.util.List` 🡸 today!
- `java.util.Set`
- `java.util.Queue`

`java.util.Map`

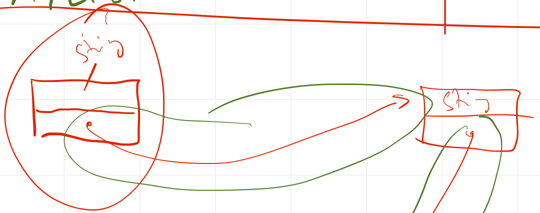
- `java.util.SortedMap`

Memory Handling

int[] iA = new int[3];

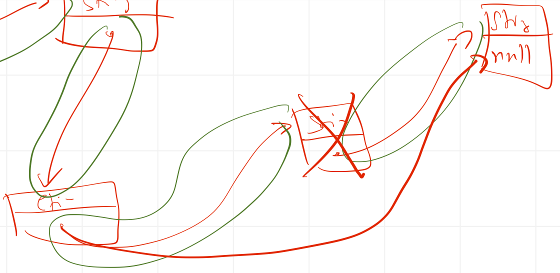


ARRAYLIST



Car c = new Car();

LINKEDLIST



List

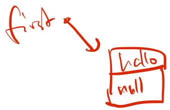
- Finite number of ordered elements, allowing duplicates
- Access via index values
- java.util.List
    - `add`, `addAll`, `set`, `replaceAll`
    - `contains`, `containsAll`, `isEmpty`, `size`
    - `remove`, `removeAll`, `clear`
    - `subList`, `iterator`, `listiterator`
    - `sort`

List

- Finite number of ordered elements, allowing duplicates
- Access via index values
- java.util.List
  - `add`, `addAll`, `set`, `replaceAll`
  - `contains`, `containsAll`, `isEmpty`, `size`
  - `remove`, `removeAll`, `clear`
  - `subList`, `iterator`, `listiterator`
  - `sort`
- Implementations
  - `java.util.ArrayList`: Uses an array internally
  - `java.util.LinkedList`: Uses a linked list internally

1. first = null | add("hallo") | ✓

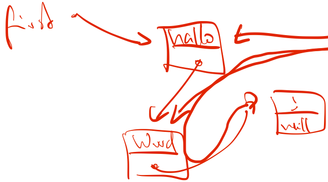
2.



last()
current
(36)

demo

3.



last()
current ✓
current ✓
current
⊖

(36)
(37)
(38)
(37)
(38)

ArrayList vs. LinkedList

```java
// ArrayList
List<Student> arr = new ArrayList<Student>(300);
// ...
arr.set(154, new Student("Maria"));
arr.set(203, new Student("Hans"));
// ...
arr.get(203).doSomething();

// LinkedList
List<Student> ll = new LinkedList<Student>();
ll.add(new Student("Maria"));
ll.add(new Student("Hans"));
// ...
ll.get(203).doSomething();
```

Speed Differences

- Many library functions hide complexity
- ⚠ This does not mean that the complexity is gone

Speed Differences

- Many library functions hide complexity
- ⚠ This does not mean that the complexity is gone

Arrays / ArrayList

- ‘constant access’: Accessing the 5th or the 9000th elements takes the same time
- Enlarging an array after creation is costly (because the entire array needs to be copied elsewhere)

Speed Differences

- Many library functions hide complexity
- ⚠ This does not mean that the complexity is gone

Arrays / ArrayList

- ‘constant access’: Accessing the 5th or the 9000th elements takes the same time
- Enlarging an array after creation is costly (because the entire array needs to be copied elsewhere)

LinkedList

- The longer the list the longer it takes to access an element
- Enlarging is constant, removal in the middle as well

demo

Exercise

https://github.com/idh-cologne-java-2/exercise-04