Recap: Generics and Lists

Generics

- Template classes usable for multiple classes
  - E.g., collections
- Syntactically denoted by < >

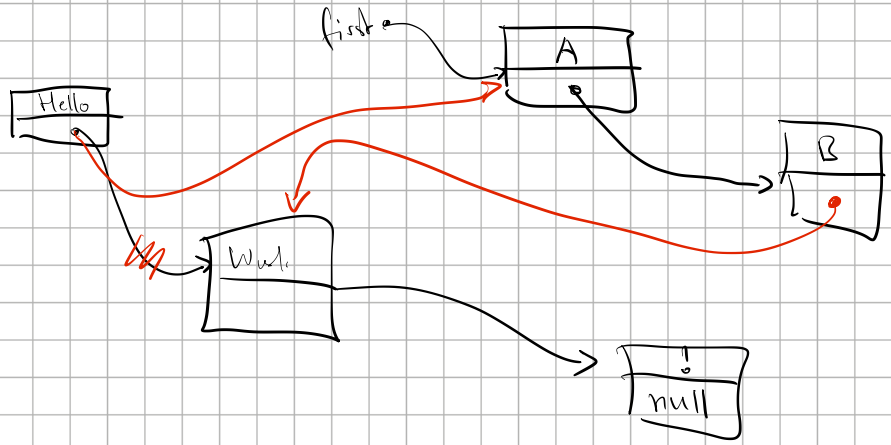
Lists

- Unidimensional, ordered collection
- Two implementations available
  - ArrayList: Uses an array internally
    - Array: Elements are stored in a continuous block of memory
  - LinkedList: Uses a linked list internally
    - List: Elements are distributed all over the place, but linked

Exercise 5

- Big Picture
  - Using the iterator solves most of the problems
  - Handling the first element is cumbersome – storing a 'prefirst' dummy element makes it easier
  - Two steps for the most complex method: `addAll(int index, Collection c)`
    - Put the elements of `c` into a linked list
    - Insert it at the right position
  - Testing all methods is tedious – automatic testing to the rescue
    - ➔ Later in the semester

addAll(int index, Collection<T> coll) || addAll(1, ["A", "B"])



Session 6: Collections, Part 2 (Queues and Sets)

Fortgeschrittene Programmierung (Java 2)

Nils Reiter
nils.reiter@uni-koeln.de

May 17, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Queue and Stack

Interfaces

`java.util.Collection`

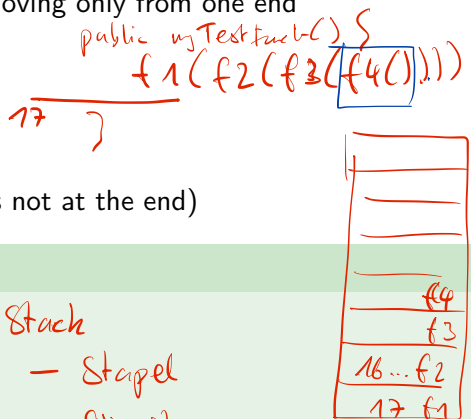
- `java.util.List` ← last week
- `java.util.Queue` ← today / Stack
- `java.util.Set` ← today

`java.util.Map`

- `java.util.SortedMap`

Queue and Stack

- Ordered collection, changeable by adding/removing only from one end
- Last In, First Out (LIFO, Stack)
  - Same end for adding and removing elements
- First In, First Out (FIFO, Queue)
  - Different end for adding and removing
- No random access (i.e., no access to elements not at the end)



Examples

Queue
- Warteschlange
- Switch-Bar
- Videospiele

Stack
- Stapel
- Stapelweise
- Downloads

Queue in Java

- Interface java.util.Queue<E>
  - Special case: capacity-restricted Queue (i.e., one with a limited size)
- Defines several methods:

	Throws exception	Returns special value
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

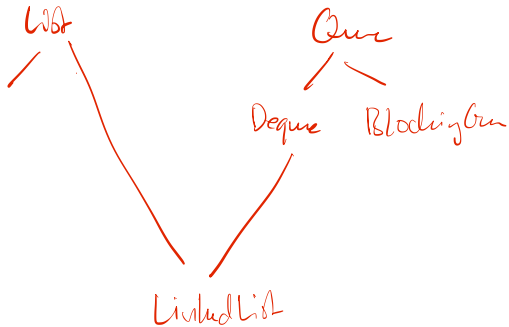
Table: Queue Methods

Queue in Java

Sub Interfaces

- java.util.Deque<E>
  - “Deque”: double ended queue
  - Access on both ends (but not in the middle)
- java.util.BlockingQueue<E> / java.util.BlockingDeque<E>
  - Wait for the queue to become non-empty when retrieving
  - Wait for space to become available in the queue when adding

Implementations



- Based on an array: java.util.ArrayDeque<E>
- Based on linked list: java.util.LinkedList<E>

demo

Section 2

Set

Sets

- Mathematical concept based on set theory

W Set_theory

$$S = \{1, 2, 3\} \qquad \{1, 2\} \cap \{2, 3\} = \{2\} \qquad \{1, 2\} \cup \{2, 3\} = \{1, 2, 3\} \qquad \emptyset = \{\}$$

Sets

- Mathematical concept based on set theory

  W Set_theory

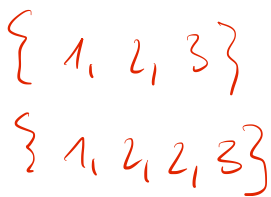
$$S = \{1, 2, 3\} \qquad \{1, 2\} \cap \{2, 3\} = \{2\} \qquad \{1, 2\} \cup \{2, 3\} = \{1, 2, 3\} \qquad \emptyset = \{\}$$

- Unordered collections, cannot contain the same thing twice
  - `add(e)` returns `false` if `e` was already in the set
- No random access to specific elements – there is no index, because there is no order
- Access only via iterators
- java.util.Set<E>

Implementation: java.util.HashSet

This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.

java.util.HashSet<E>



When are Two Objects 'the Same'?

- Every object inherits from java.lang.Object
- Two important methods: `hashCode()` and `equals(Object o)`

When are Two Objects ‘the Same’?

- Every object inherits from java.lang.Object
- Two important methods: `hashCode()` and `equals(Object o)`

```
boolean equals(Object o)
```

- Reflexive, symmetric, transitive, consistent
- `x.equals(null)` is `false` for any object x

When are Two Objects 'the Same'?

- Every object inherits from java.lang.Object
- Two important methods: `hashCode()` and `equals(Object o)`

```
boolean equals(Object o)
```

- Reflexive, symmetric, transitive, consistent
- `x.equals(null)` is `false` for any object x

```
int hashCode()
```

- If `x.equals(y)` returns `true`, `x.hashCode() == y.hashCode()`
- Used extensively in collections

`equals()` vs. `==`

- `==` compares if the objects are the same
  - I.e.: If they refer to the same unit in memory
- `equals()` lets the objects decide their equality
  - By overwriting the method in a class
  - By default (java.lang.Object.equals()): `x.equals(y)` is `true` iff `x==y`

demo

Exercise

https://github.com/idh-cologne-java-2-summer-2023/exercise-06