Secure OOP with Java

Lecture Unit - 09

Claudia Maderthaner <claudia.maderthaner@fh-hagenberg.at>

Data Structures

Data Structures

- Organization of data
- Related data and operations on that data

Data Structure Operations

- Read
- Search
- Insert
- Delete

Classifications

- Linear vs. non-linear
- Static vs. dynamic
- Homogenous vs non-homogenous

Abstract Data Types

Abstract Data Types

- List
  - Stack
  - Queue
  - Tuple
  - Stream
- Set
- Map
- Graph
  - Tree

List

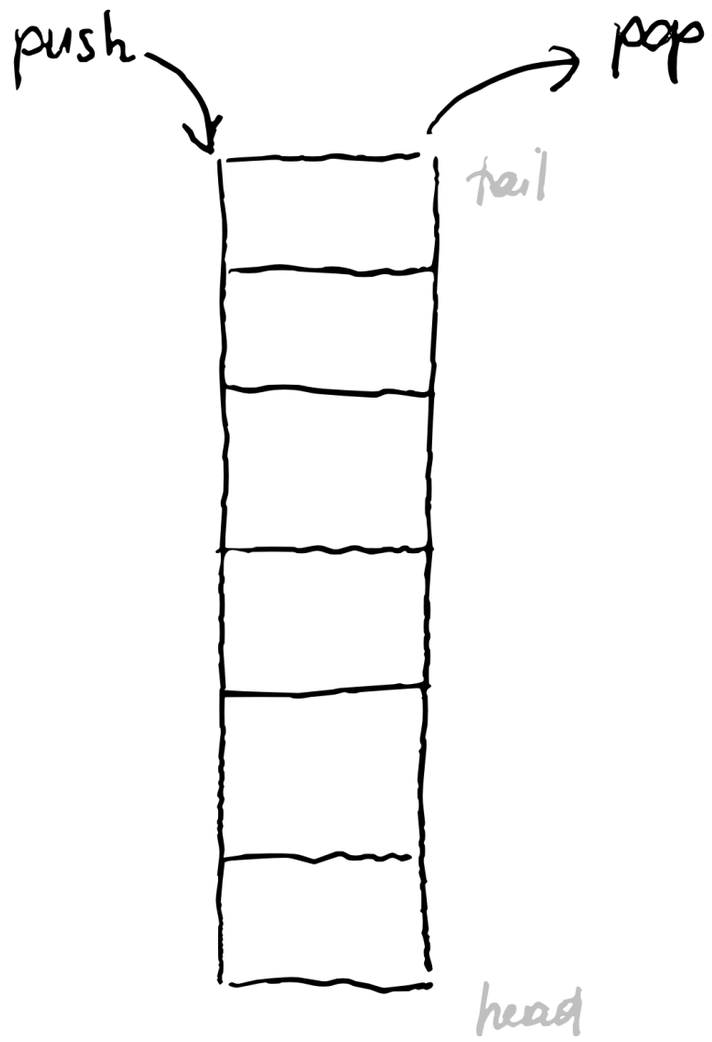
- Finite number of ordered values
- Duplicates are allowed

Stack

last in, first out (LIFO)

Stack

- Finite number of ordered values
- Duplicates are allowed
- Operations
  - **`push`** - add element to stack
  - **`pop`** - remove and return most recent element from stack
  - `peek` - return most recent element without removing it
  - `empty` - returns `boolean` value denoting whether the stack has any elements
  - `count` - returns the total number of elements currently in the stack
  - `clear` - remove all objects from stack

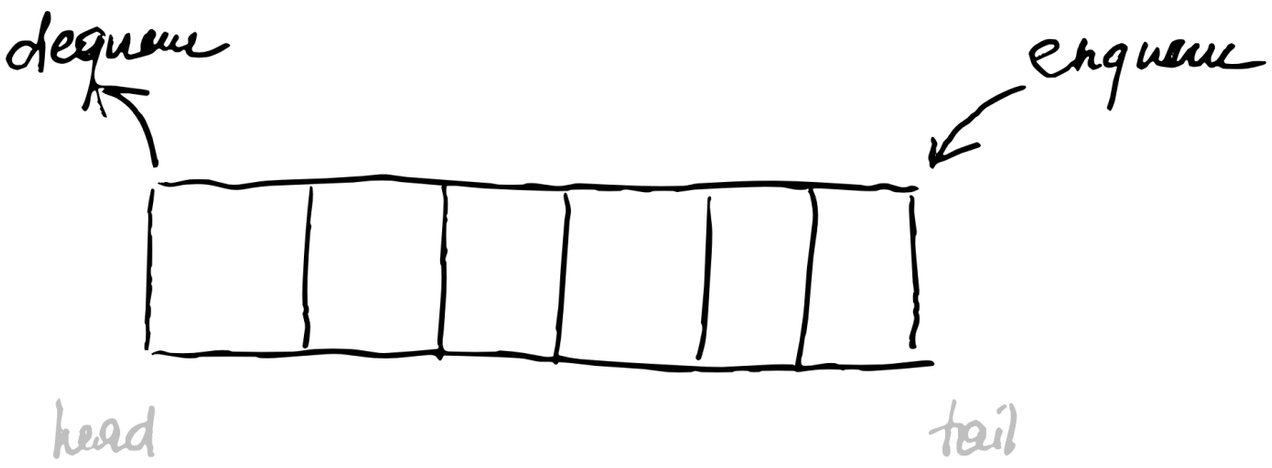

push
pop
tail
head

Queue

first in, first out (FIFO)

Queue

- Finite number of ordered values
- Duplicates are allowed
- Operations
  - **`enqueue`** - add element at end of queue
  - **`dequeue`** - remove and return the first element of the queue
  - `peek` - return first element without removing it
  - `empty` - returns `boolean` value denoting whether the queue has any elements
  - `count` - returns the total number of elements currently in the queue
  - `clear` - remove all objects from queue


dequeue
enqueue
head
tail

Tuple

- Specific number of ordered values
- A tuple may contain duplicates
- Examples: Monuple, couple/pair, triple, quadruple, ...

Stream

- (Potentially) infinite number of ordered values
- Duplicates are allowed

Set

- Finite number of elements without particular order
- No duplicates

Map

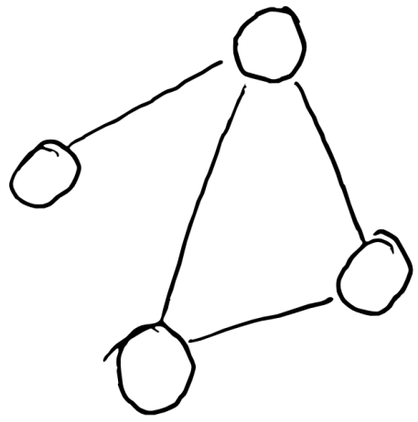
aka Associative Array, Dictionary

- Finite Collection of key/value pairs
- Each key appears at most once in the collection
- Operations
  - `add/insert` - add new key/value pair
  - `get` - lookup value by key
  - `update` - update value for already existing key
  - `remove/delete` - remove key/value pair given a valid key
  - `contains` - returns `boolean` value denoting whether a key is present

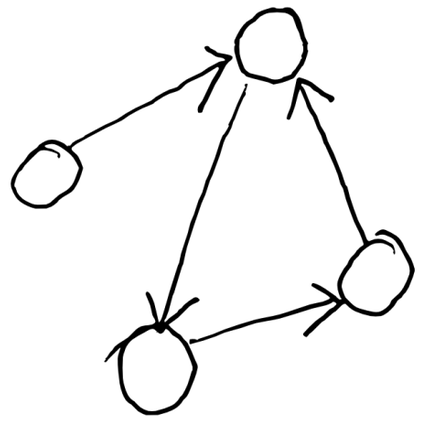
Graph

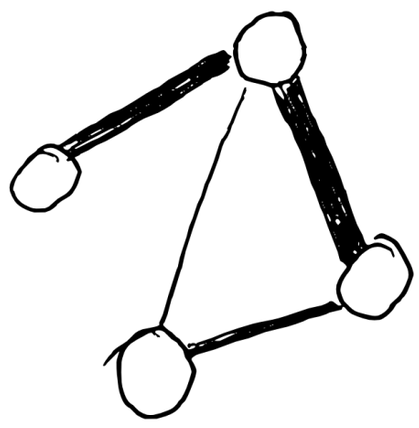
- Finite set of nodes (vertices) together with a set of pairs of this nodes (edges)
  - Undirected graph - unordered pairs of nodes
  - Directed graph - ordered pairs of nodes
  - Weighted graph - pairs are associated with a value

undirected



directed



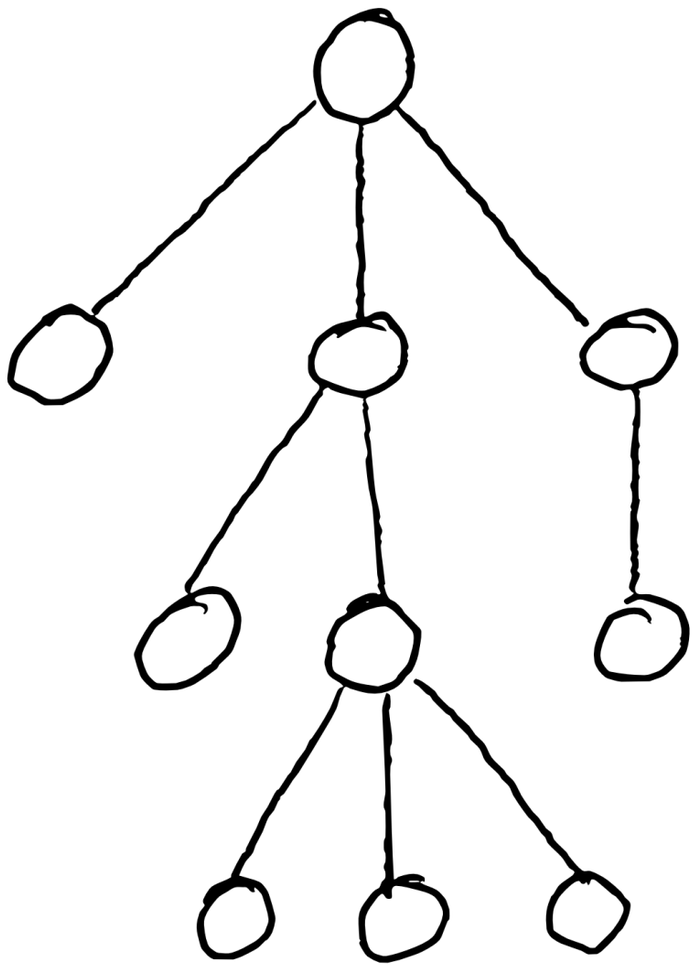


weighted

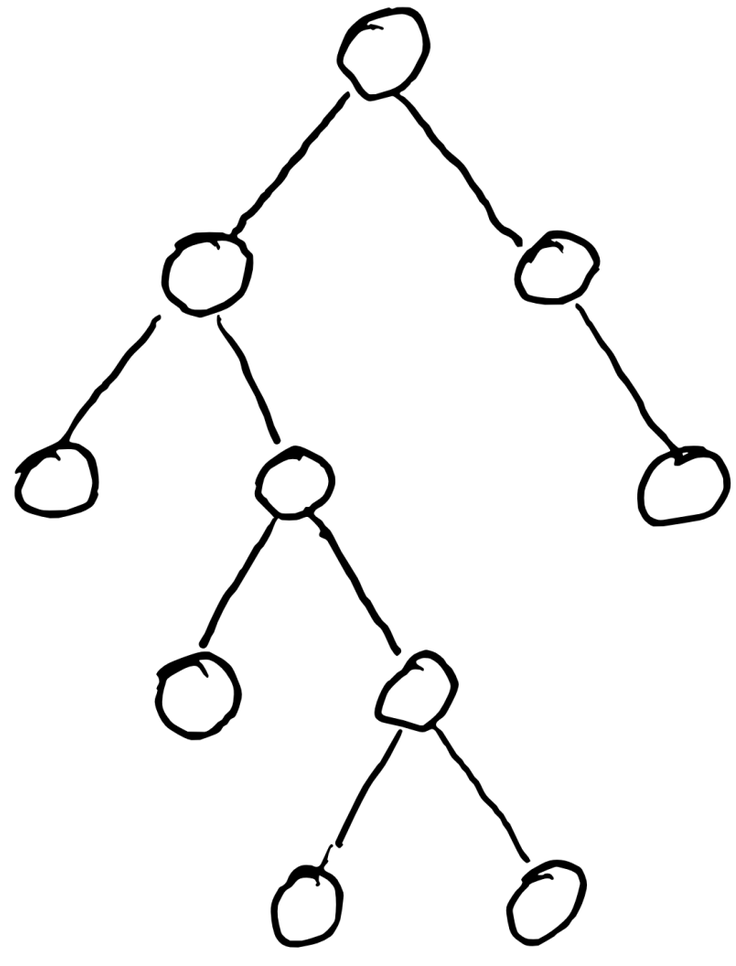
Tree

- Finite set of connected nodes in a hierarchical tree structure
- Every node (except for the root node) must be connected to exactly one parent (no cicles or loops)

Tree



Binary Tree



Arrays

Arrays

- Ordered collection of elements
- The type of elements is called the **base type**
- The number of elements it holds is a fixed attribute called `length`

Accessing Array Elements

```
int[] a = new int[10] // => { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
for (int i = 0; i < a.length; i++) {
    a[i] = i;
}
```

Array Bounds

```
int[] a = new int[3]

a[0] = 1;
a[1] = 2;
a[3] = 3; //  Exception java.lang.ArrayIndexOutOfBoundsException
```

Types of Array

- Primitive Arrays

```
int[] numbers = new int[] { 1, 2, 3 };
```

- Object Arrays

```
String[] names = new String[] { "John", "Paul", "Ringo", "George" };
```

- Mixed Arrays

```
Object[] things = new Object[] { 1, "Paul", null };
```

- Multidimensional Arrays

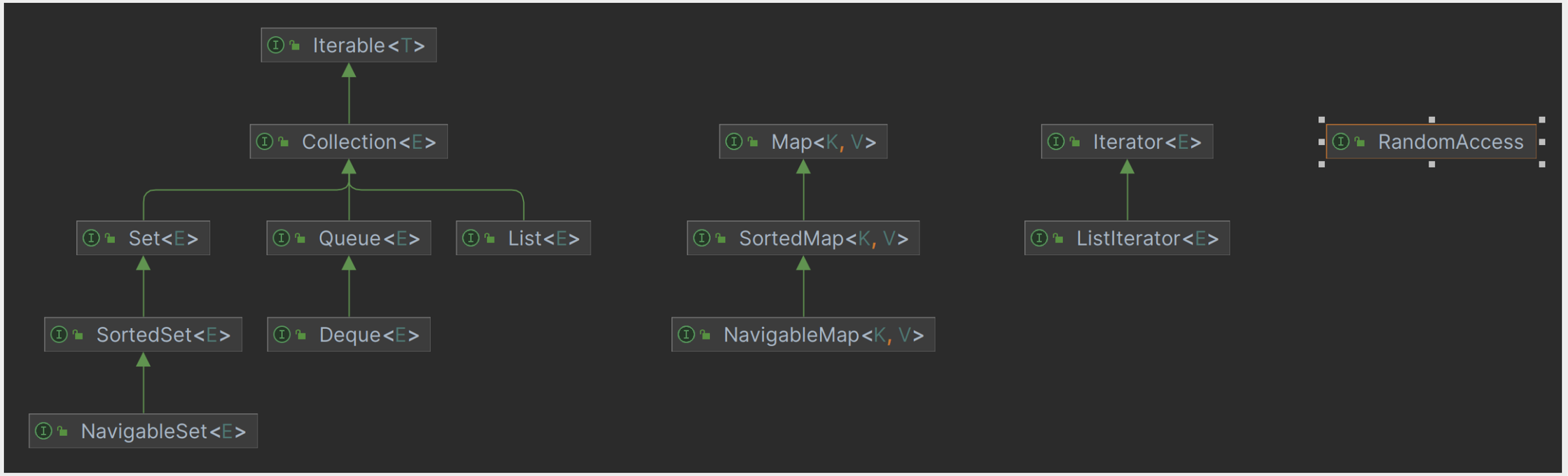
```
int [][] twoDimensions = new int[5][5];
twoDimensions[0][0] = 0;
twoDimensions[0][1] = 1;
twoDimensions[4][4] = 100;
int value = twoDimensions[1][1];
```

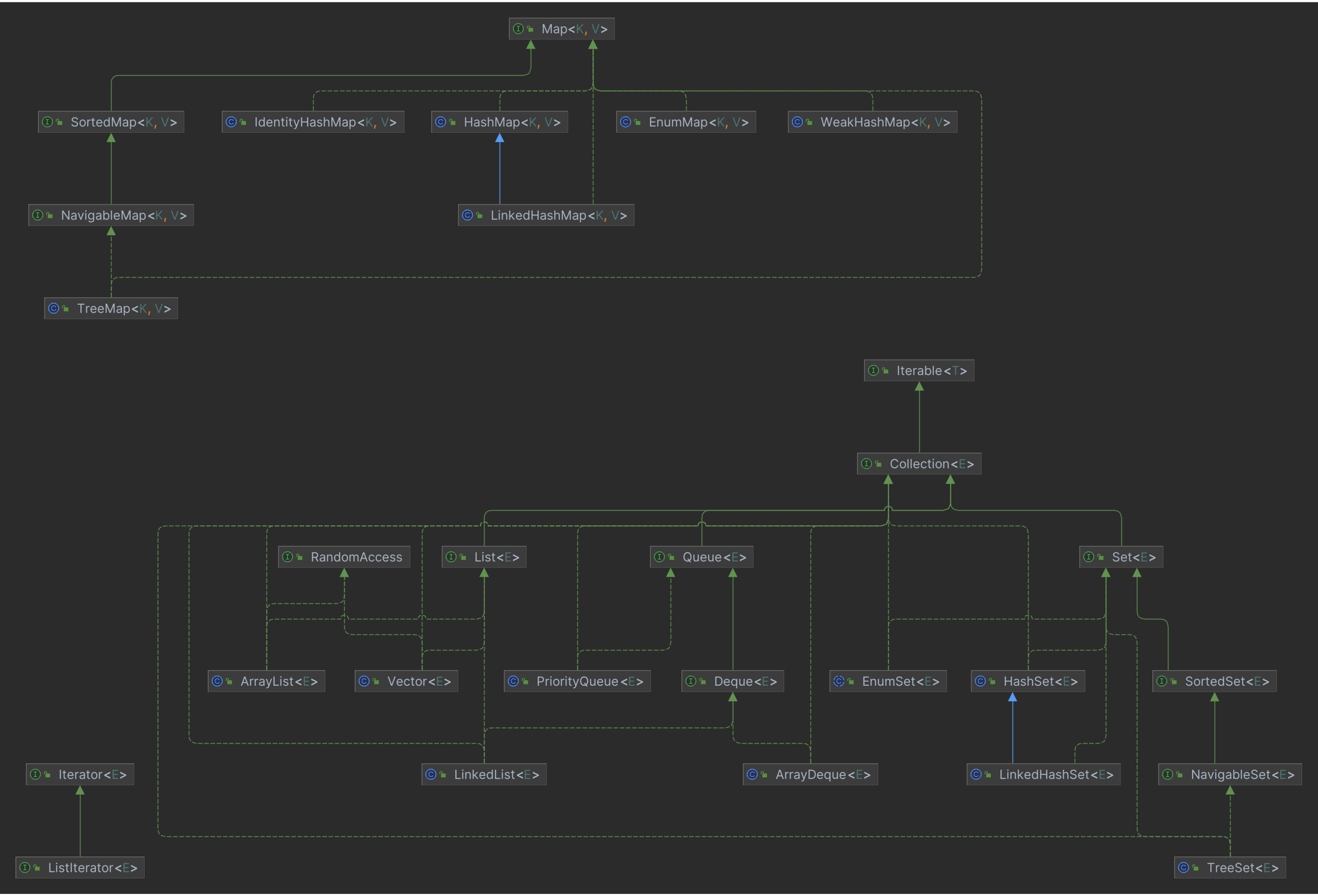
```
int [][][] threeDimensions = new int[2][2][3]

threeDimensions[1][1][1] = 20;
threeDimensions[1][1][2] = 30;
int value = threeDimensions[1][1][1];
```

Java Collection Framework

Interfaces

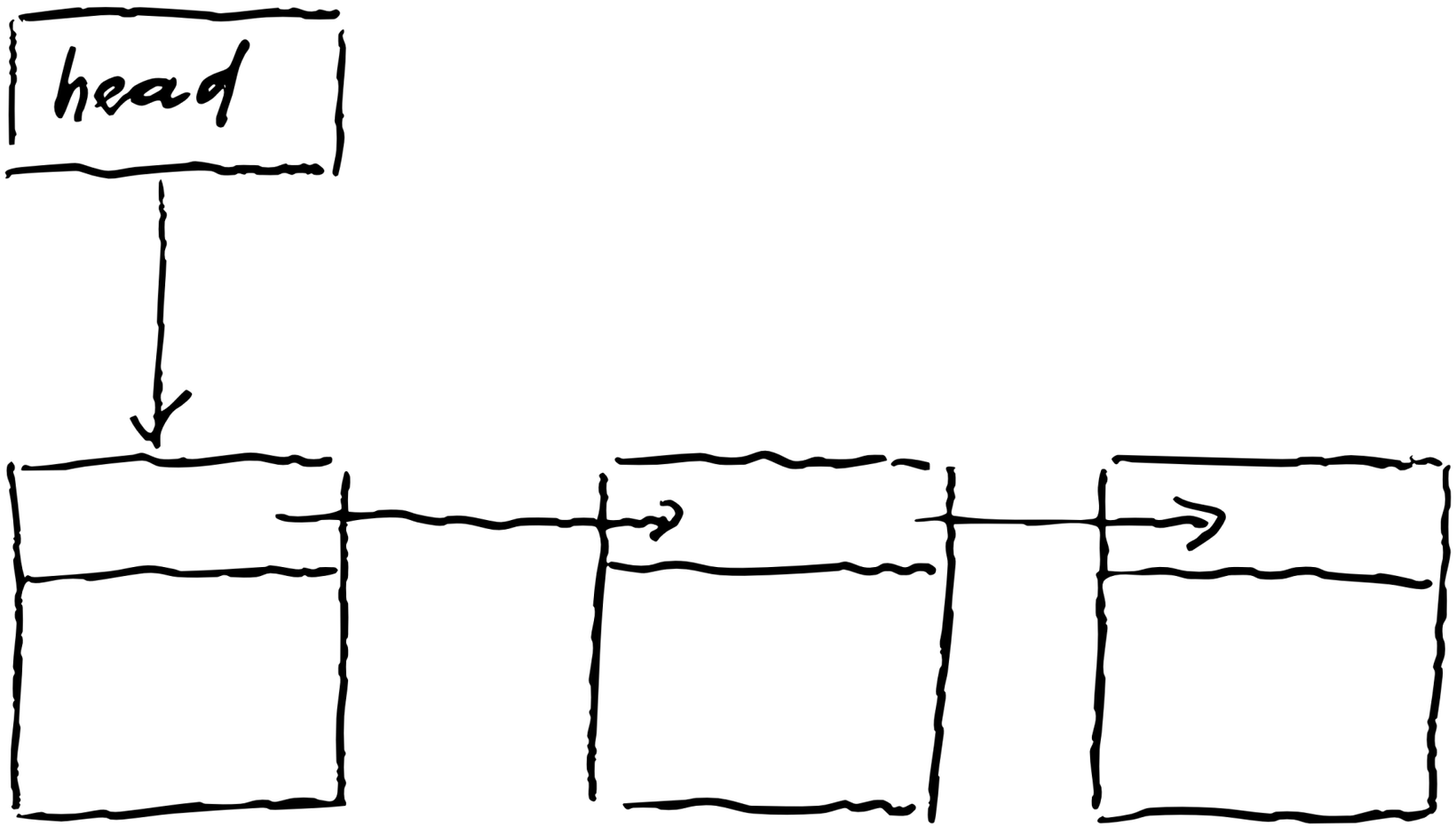

Iterable<T>
Collection<E>
Set<E>
Queue<E>
List<E>
SortedSet<E>
Deque<E>
NavigableSet<E>
Map<K, V>
SortedMap<K, V>
NavigableMap<K, V>
Iterator<E>
ListIterator<E>
RandomAccess


Map<K, V>
SortedMap<K, V>
IdentityHashMap<K, V>
HashMap<K, V>
EnumMap<K, V>
WeakHashMap<K, V>
NavigableMap<K, V>
LinkedHashMap<K, V>
TreeMap<K, V>
Iterable<T>
Collection<E>
RandomAccess
List<E>
Queue<E>
Set<E>
ArrayList<E>
Vector<E>
PriorityQueue<E>
Deque<E>
EnumSet<E>
HashSet<E>
SortedSet<E>
Iterator<E>
LinkedList<E>
ArrayDeque<E>
LinkedHashSet<E>
NavigableSet<E>
ListIterator<E>
TreeSet<E>

Node-based Structures

Linked List

- Dynamic size
- Ease of insertion/deletion
- No random access


head

Contact

Moodle Discussion Board

claudia.maderthaner@fh-hagenberg.at