Java

Sets & Maps

Tobias Hanf, Manik Khurana

13. Dezember 2021

Java-Course

Overview

- Recap
  - Last class' exercise - Collections
  - Hands-on Exercise
- Set
- Map

Recap

Last class' exercise - Collections

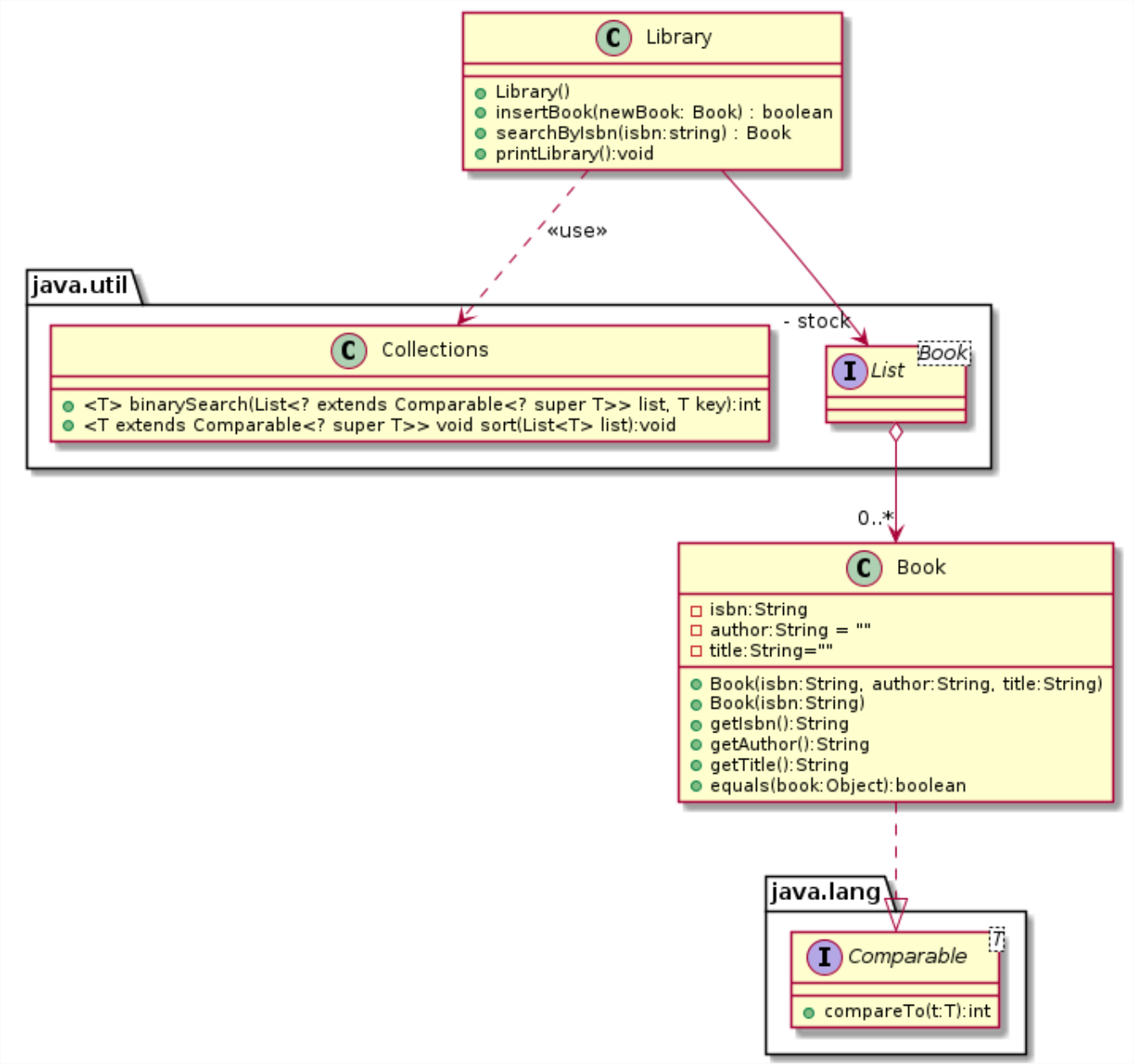
"This class consists exclusively of static methods that operate on or return collections"[1]

Some methods:

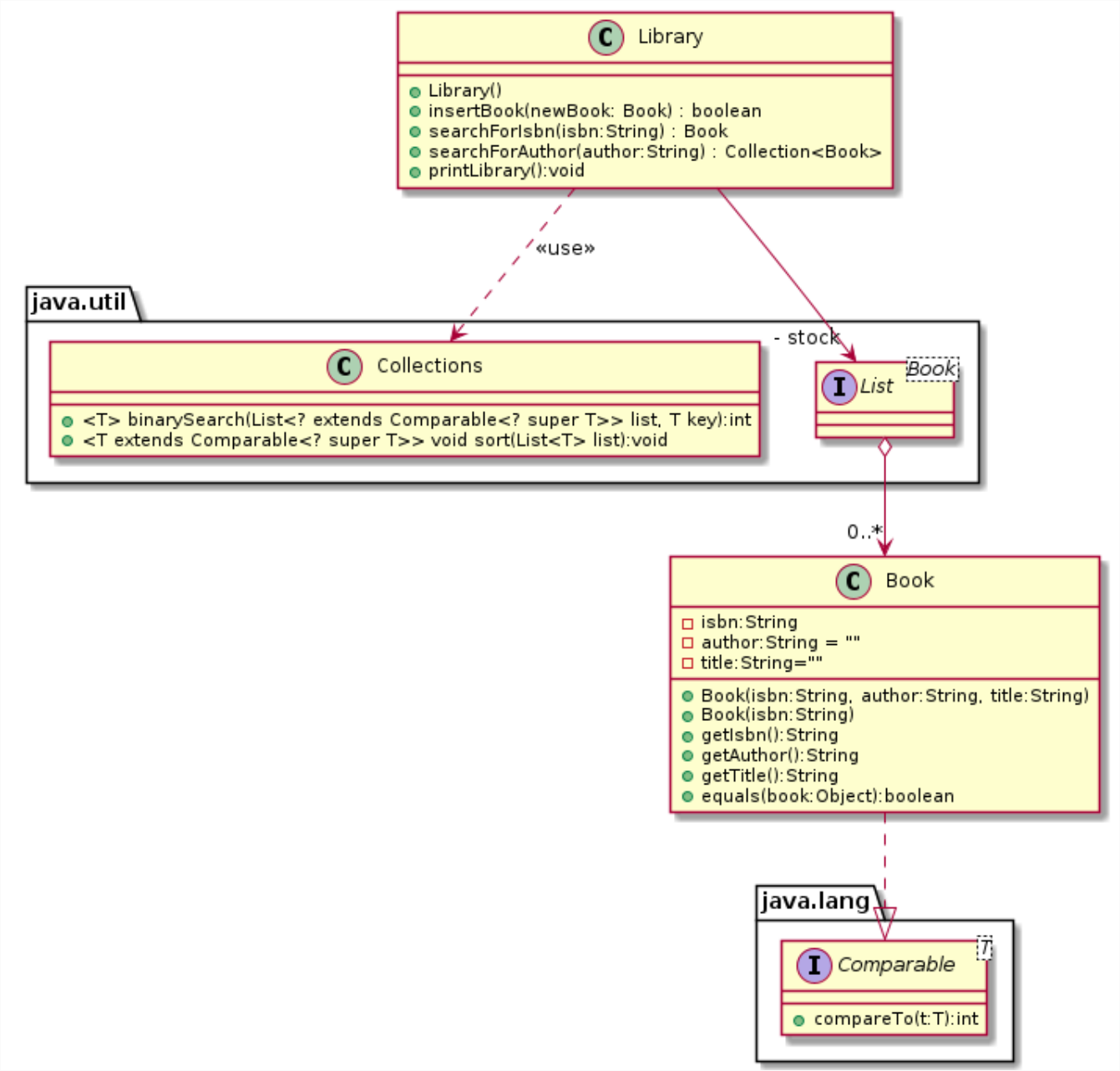
- `binarySearch(...)`
- `max(...)`
- `min(...)`
- `reverse(...)`
- `sort(...)`

[1] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html`

Part 3 - Contd. from last class



Part 4- Contd. from last class



Set

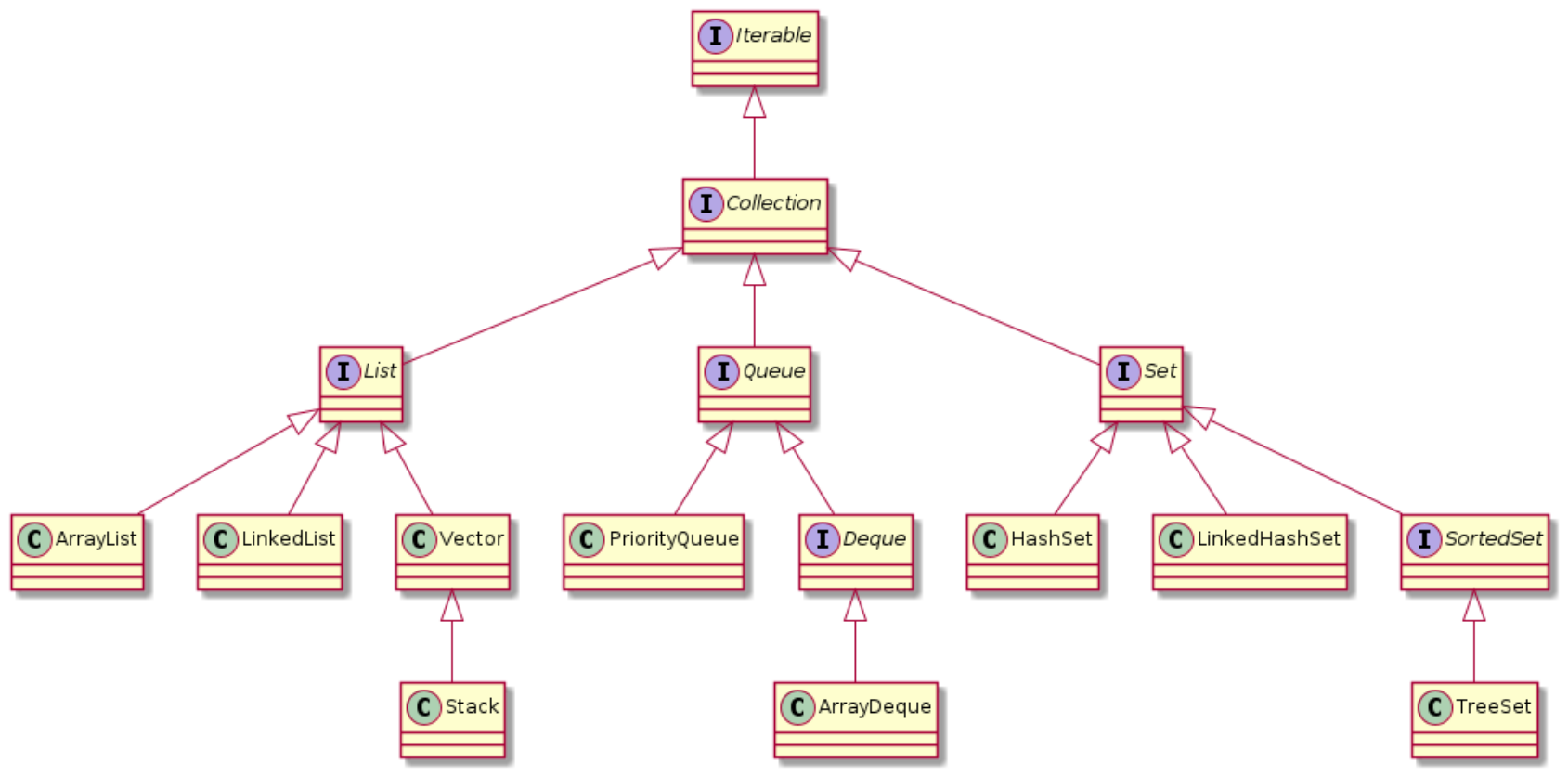
Collections Framework

Java offers various data structures like **Sets**, **Lists** and **Maps**. Those structures are part of the collections framework.

There are interfaces to access the data structures in an easy way. There are multiple implementations for various needs. Alternatively you can use your own implementations.

Documentation: `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collection.html`

Collections Overview



Set

The set interface is present in java.util package and extends the Collection interface is an unordered collection of objects in which duplicate values cannot be stored.

The Set interface is declared as:

```
1          public interface Set extends Collection
2
```

The Set object can be created as:

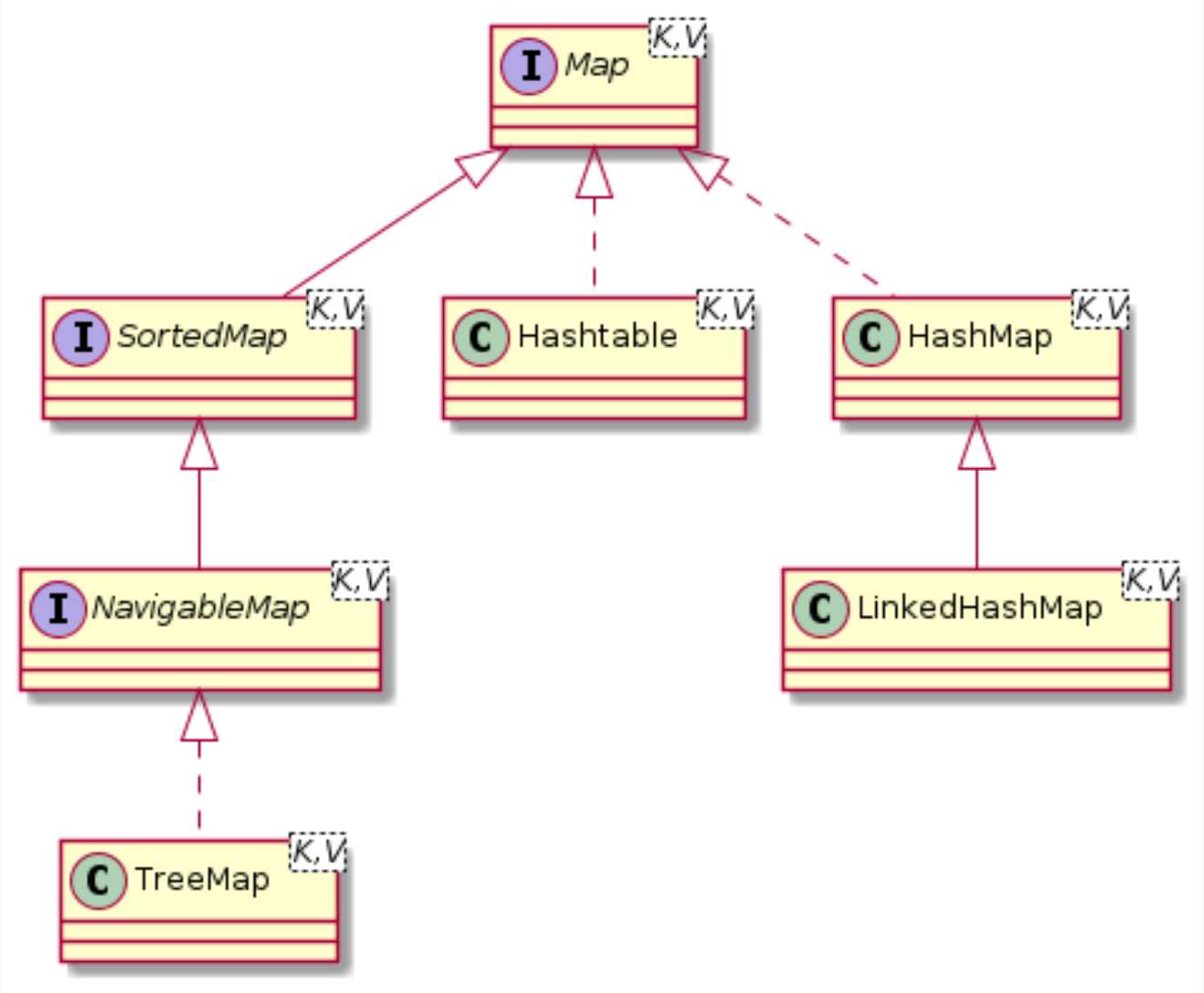
```
1          // Obj is the type of the object to be stored in Set
2          Set<Obj> set = new HashSet<Obj> ();
3
```

Set Methods

some useful Set methods:

boolean	`add(E element)`	insert element if not already prese
boolean	`contains(Object o)`	Returns true if the specified element is prese
int	`size()`	Returns the number of elements in the s

Map


Map
K,V
SortedMap
K,V
Hashtable
K,V
HashMap
K,V
NavigableMap
K,V
LinkedHashMap
K,V
TreeMap
K,V

Map

The interface *Map* is not a subinterface of *Collection*.
A map contains pairs of key and value. Each key refers to a value. Two keys can refer to the same value. There are not two equal keys in one map. *Map* is part of the package `java.util`.

```
1           public static void main (String[] args) {
2
3               Map<Integer, String> map =
4               new HashMap<Integer, String>();
5
6               map.put(23, "foo");
7               map.put(28, "foo");
8               map.put(31, "bar");
9               map.put(23, "bar"); // "bar" replaces "foo" for key = 23
10
11              System.out.println(map);
12              // prints: {23=bar, 28=foo, 31=bar}
13          }
14
```

Key, Set and Values

You can get the set of keys from the map. Because one value can exist multiple times a collection is used for the values.

```
1            public static void main (String[] args) {
2
3                // [...] map like previous slide
4
5                Set<Integer> keys = map.keySet();
6                Collection<String> values = map.values();
7
8                System.out.println(keys);
9                // prints: [23, 28, 31]
10
11               System.out.println(values);
12               // prints: [bar, foo, bar]
13           }
14
```

Nested Maps

Nested maps offer storage with key pairs.

```
1            public static void main (String[] args) {
2
3                Map<String, Map<Integer, String>> addresses =
4                new HashMap<String, Map<Integer, String>>();
5
6                addresses.put("Noethnitzer Str.",
7                new HashMap<Integer, String>());
8
9                addresses.get("Noethnitzer Str.").
10               put(46, "Andreas-Pfitzmann-Bau");
11               addresses.get("Noethnitzer Str.").
12               put(44, "Fraunhofer IWU");
13           }
14
```

Maps and Lambda

```
1            map.forEach((k,v) -> {
2                //Key and Value
3                System.out.println("Key: " + k + ", value: " v);
4            })
5
```

Maps and For Each

You can interate through the entry set of a map (available before Java 1.8)

```
Map<String, String> map = ...
for (Map.Entry<String, String> entry : map.entrySet()) {
    System.out.println("Key: " + entry.getKey() +
    ", value" + entry.getValue());
}

```

Map Methods

some useful Map methods:

`V get(Object key)`

Returns the value to which the specified key is mapped.

`V remove(Object key)`

Removes the mapping for a key from this map if it is present

`V put(K key, V value)`

Associates the specified value with the specified key in this map

`Collection<V> values()`

Returns a collection view of the values contained in the map

`Set<K> keySet()`

Returns a set view of the keys contained in the map

TreeMap vs HashMap i

TreeMap[2]:

- Red-Black tree implementation
- has an ordering $\rightarrow$ can be sorted
- guaranteed `log(n)` time constant for `get`, `put` and `remove`

TreeMap vs HashMap ii

HashMap[3]:

- Hash table implementation
- mostly constant time for `get` and `put`
- *initial capacity* and *load factor* determine performance

[2] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/TreeMap.html`
[3] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html`

Overview

List	Keeps order of objects Easily traversible Search not effective
Set	No duplicates No order - still traversible Effective searching
Map	Key-Value storage Search super-effective Traversing difficult