Java

Collections

Tobias Hanf, Manik Khurana

13. Dezember 2021

Java-Course

Overview

Recap

Last class' exercise - Collections

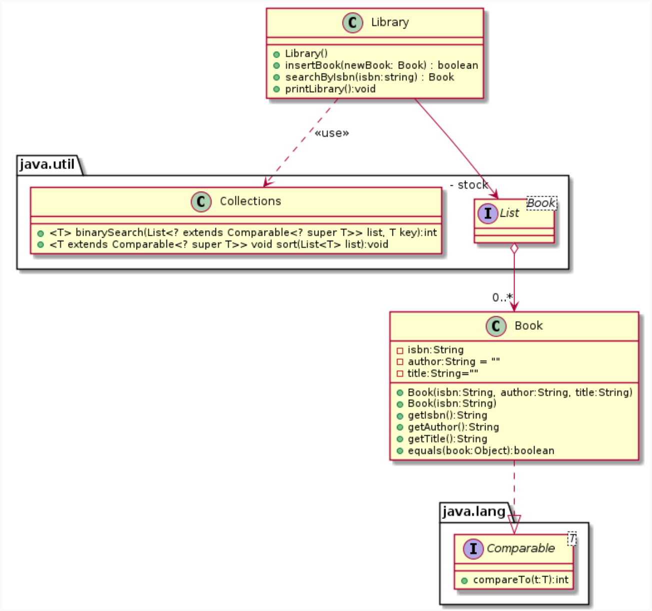
"This class consists exclusively of static methods that operate on or return collections"[1]

Some methods:

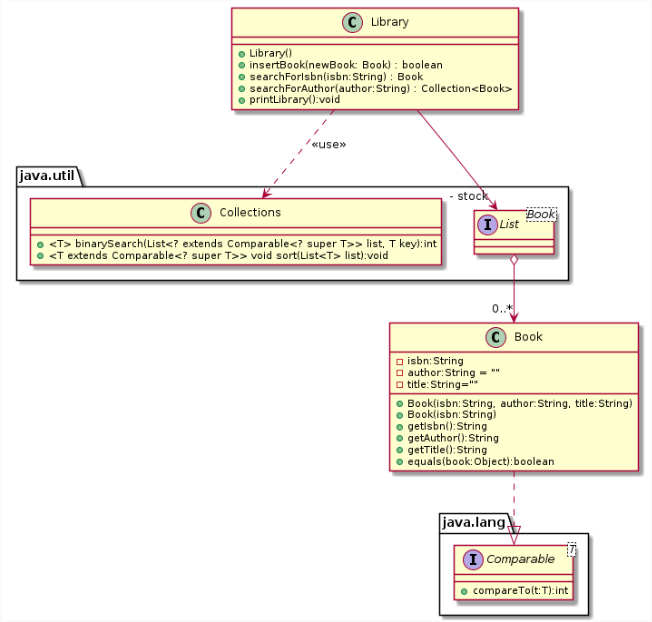
- `binarySearch(...)`
- `max(...)`
- `min(...)`
- `reverse(...)`
- `sort(...)`

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

Part 3 - Contd. from last class


Library
Library()
insertBook(newBook: Book) : boolean
searchByIsbn(isbn:string) : Book
printLibrary():void
«use»
java.util
Collections
<T> binarySearch(List<? extends Comparable<? super T>> list, T key):int
<T extends Comparable<? super T>> void sort(List<T> list):void
- stock
List
Book
0..*
Book
isbn:String
author:String = ""
title:String=""
Book(isbn:String, author:String, title:String)
Book(isbn:String)
getIsbn():String
getAuthor():String
getTitle():String
equals(book:Object):boolean
java.lang
Comparable
T
compareTo(t:T):int

Part 4- Contd. from last class



Library
Library()
insertBook(newBook: Book) : boolean
searchForIsbn(isbn:String) : Book
searchForAuthor(author:String) : Collection<Book>
printLibrary():void
«use»
java.util
Collections
<T> binarySearch(List<? extends Comparable<? super T>> list, T key):int
<T extends Comparable<? super T>> void sort(List<T> list):void
- stock
List
Book
0..*
Book
isbn:String
author:String = ""
title:String=""
Book(isbn:String, author:String, title:String)
Book(isbn:String)
getIsbn():String
getAuthor():String
getTitle():String
equals(book:Object):boolean
java.lang
Comparable
T
compareTo(t:T):int

Set

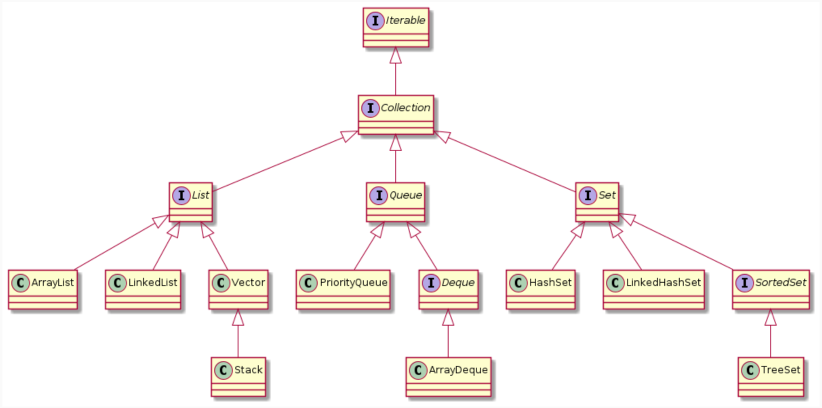
Collections Framework

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:

```java
            public interface Set extends Collection

```

The Set object can be created as:

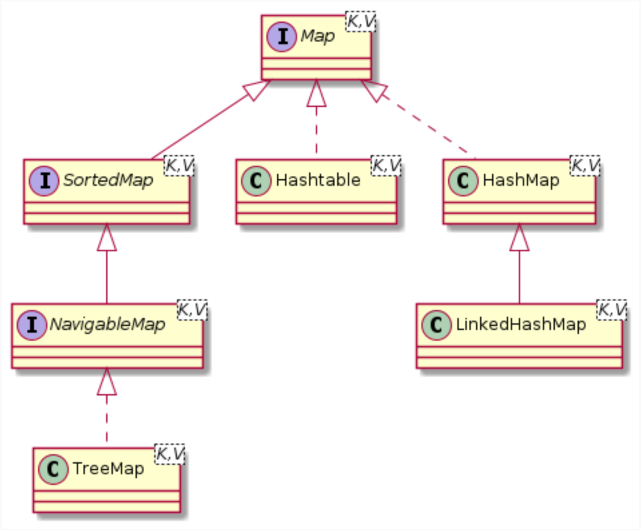
```java
            // Obj is the type of the object to be stored in Set
            Set<Obj> set = new HashSet<Obj> ();

```

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`.

```java
        public static void main (String[] args) {

            Map<Integer, String> map =
            new HashMap<Integer, String>();

            map.put(23, "foo");
            map.put(28, "foo");
            map.put(31, "bar");
            map.put(23, "bar"); // "bar" replaces "foo" for key = 23

            System.out.println(map);
            // prints: {23=bar, 28=foo, 31=bar}
        }

```

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.

```
public static void main (String[] args) {

    // [...] map like previous slide

    Set<Integer> keys = map.keySet();
    Collection<String> values = map.values();

    System.out.println(keys);
    // prints: [23, 28, 31]

    System.out.println(values);
    // prints: [bar, foo, bar]
}

```

Nested Maps

Nested maps offer storage with key pairs.

```java
public static void main (String[] args) {

    Map<String, Map<Integer, String>> addresses =
    new HashMap<String, Map<Integer, String>>();

    addresses.put("Noethnitzer Str.",
    new HashMap<Integer, String>());

    addresses.get("Noethnitzer Str.").
    put(46, "Andreas-Pfitzmann-Bau");
    addresses.get("Noethnitzer Str.").
    put(44, "Fraunhofer IWU");
}

```

Maps and Lambda

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

```

Maps and For Each

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

```java
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