Collection Framework

Tobias Hanf, Maik Göken

January 30, 2023

Learn Programming with Java

Outline

Revision

Quiz

https://pingo.coactum.de/580337

Generic Programming

Very Short Introduction

In generic programming we view data types just as another parameter for your functions (classes). It allows us to write code which can work with a wide range of data types without the need for several custom implementations.

Generic Programming in Java

Java has two types of Generics:

Generic Methods:

- Generic Types per Method

Generic Classes:

- Generic Types per Class

Generic Classes

A programmer can provide a specific type for **T1** to **Tn**. These types must be **Classes**.

We can only use methods that every Class provides.

```
1 class name<T1, T2, ..., Tn> { /* ... */ }
```

Example

```
1  public class Tuple<T> {
2      private T item1 = null;
3      private T item2 = null;
4
5      public void set(T item1, T item2) {
6      this.item1 = item1;
7      this.item2 = item2;
8      }
9
10     public T get(int index) {
11     if( index == 0 )
12         return item1;
13     else
14         return item2;
15     }
16 }
```

Example

Now we can create vectors for different data types.

```
1 Tuple<Integer> t1 = new Tuple<>();
2 t1.set(1, 11);
3
4 Tuple<Double> t2 = new Tuple<>();
5 t2.set(1, 22.222);
```

Wrapper Classes

Because we cannot use primitive data types for generics, so we have to use wrapper classes, wrapping primitive data types inside classes. They are already part of Java.

boolean	Boolean
byte	Byte
char	Character
int	Integer
float	Float
double	Double
long	Long
short	Short

The Collection Framework

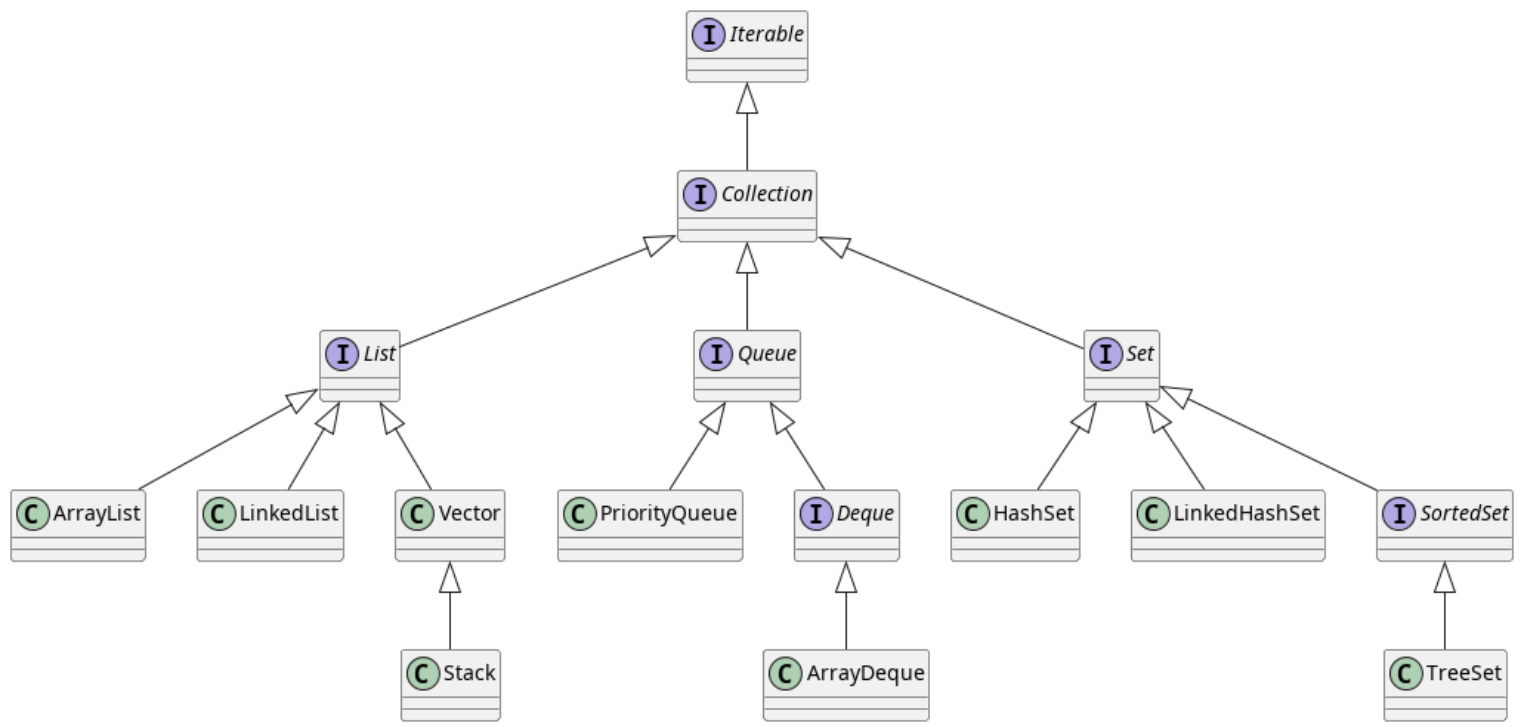
About the Framework

The Collection Framework is a set of interfaces and implementation of various data structures providing a collection objects (container) for storing ”arbitrary” data types.

It simplifies writing code in Java because it provides useful data structures such as Lists, Sets and Maps which can grow dynamically.

Every specialization of the `Collection` interfaces only adds functionality.

The hierarchy



Java Doc:

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

Big 3

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

List

List Interface

A `List` represents an ordered collection. Every item in the collection has a well defined position.

```
List<E> list;
```

boolean	`add(E e)`	append element to the end
void	`add(int index, E element)`	insert element at position index
E	`get(int index)`	get element at position index
E	`set(int index, E element)`	replace element at position index
E	`remove(int index)`	remove element at position index

Implementations

`ArrayList` [1]:

- Resizeable-array implementation
- Must resize if capacity it to low
- Else like normal array

`LinkedList` [2]:

- Doubly-linked List implementation
- Can grow without resizing
- Operation speed depending on position

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

Example

```
1  List<String> list1 = new LinkedList();
2
3  list1.add("Hello");
4  list1.add(1, ",");
5  list1.add("World!");
6
7  System.out.println(list1.get(2));
8
9  list1.set(2, "Dresden");
10
11 System.out.println(list1.get(2));
```

For-Each Loop

Classes who implement the `Iterator`[3] interface can be used in a For-Each loop:

```
1
2 List<String> list1 = new LinkedList();
3
4 // For-Each Loop
5 for(String s : list1) {
6     System.out.print(s);
7 }
```

This will iterate over every element inside the `List`.

[3] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Iterator.html

Set

Set Interface

A `Set` is an unordered collection which cannot store duplicate objects.

```
Set<E> set = new $SetImplementation$<E>();
```

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

Implementations

- `HashSet`
- `LinkedHashSet`
- `TreeSet`

Example

```
1 Set<Integer> set1 = new HashSet<Integer>();
2
3 set1.add(1);
4 set1.add(2);
5 set1.add(1);
6
7 // Returns 2
8 set1.size();
```

Map

Map Interface

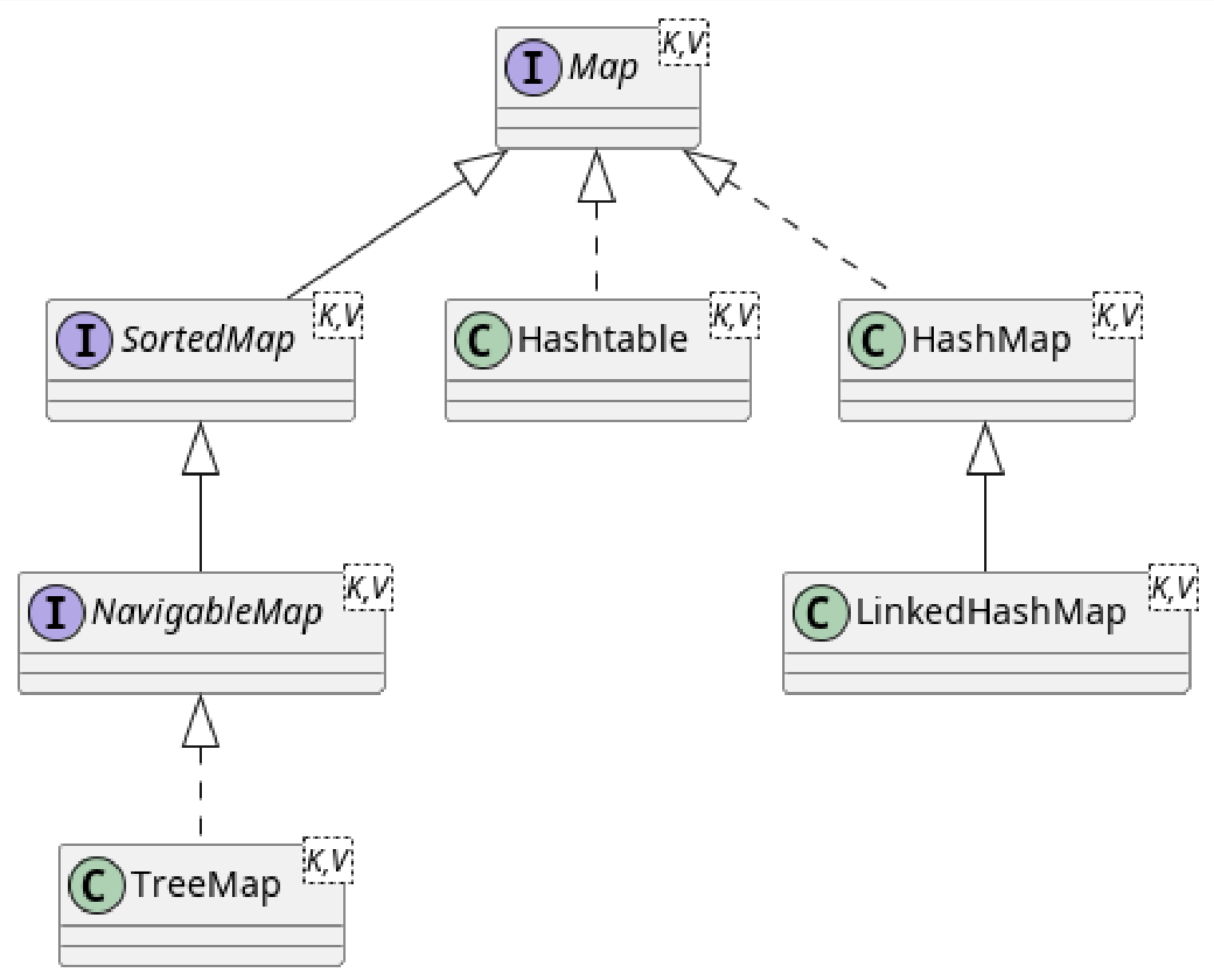
The `Map` interface is not a collection. It contains pairs of keys and values. Each key references a value. Two keys can reference the same value but there can not be two equal keys.

```
Map<K, V> map = HashMap<K,V>();
```

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

Implementations



Example

```
1 Map<Integer, String> map = new HashMap<Integer, String
      >();
2
3 map.put(23, "foo");
4 map.put(28, "foo");
5 map.put(31, "bar");
6 map.put(23, "bar"); // "bar" replaces "foo" for key = 23
7
8 System.out.println(map);
9 // prints: {23=bar, 28=foo, 31=bar}
```

Learning Resources

Learning Resources

Some websites you may want look at if you want to known more.

Generics:

`https://docs.oracle.com/javase/tutorial/java/generics/index.html`

Collection Framework:

`https://docs.oracle.com/javase/tutorial/collections/TOC.html`

`https://www.w3schools.blog/collection-tutorial-java`

Exercise

URP V3.1

Improve the Version 3 of the University Resource Planner by replacing all arrays with more appropriate data structures. Also the storage of students should allow the retrieval of student by providing their enrollment number.

Discussion

Discuss which dynamic data structures could be used for each use case.

Implementation

Implement the discussed Changes.