Java

Collections

Julius Felchow (Mail - julius.felchow@mailbox.tu-dresden.de),
Benjamin Weller (Mail - benjamin.weller@tu-dresden.de)

20. Januar 2020

Java-Kurs

Overview

Collections

Collections Framework

Java offers various data structures like **Lists**, **Sets** 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 (e.g. Linked**List**, Array**List**).

You can also make your own implementation!

List

A list is an ordered collection.

```
1     public static void main(String[] args) {
2
3         List<String> list = new LinkedList<String>();
4
5         list.add("foo");
6         list.add("foo"); // insert "foo" at the end
7         list.add("bar");
8         list.add("foo");
9         list.remove("foo"); // removes the first "foo"
10        list.remove(1); // removes what?
11
12        System.out.println(list); // prints what?
13    }
14
```

List

A list is an ordered collection.

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

        List<String> list = new LinkedList<String>();

        list.add("foo");
        list.add("foo"); // insert "foo" at the end
        list.add("bar");
        list.add("foo");
        list.remove("foo"); // removes the first "foo"
        list.remove(1); // removes "bar"

        System.out.println(list); // prints ["foo", "foo"]
    }

```

List Methods

some useful List methods:

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

some useful LinkedList methods:

void	`addFirst(E element)`	append element to the beginning
E	`getFirst()`	get first element
void	`addLast(E element)`	append element to the end
E	`getLast()`	get last element

For Loop

The for loop can iterate over every element of a collection:

`for (E e : collection)`

```
1   public static void main(String[] args) {
2
3       List<Integer> list =
4           new LinkedList<Integer>();
5
6       list.add(1);
7       list.add(3);
8       list.add(3);
9       list.add(7);
10
11      for (Integer i : list) {
12          System.out.print(i + " "); // prints: 1 3 3 7
13      }
14  }
15
```

Set

A set is a collection that holds one type of objects. A set can not contain one element twice. Like all collections the interface *Set* is part of the package `java.util`.

```
1   import java.util.*;
2
3   public class TestSet {
4
5       public static void main(String[] args) {
6           Set<String> set = new HashSet<String>();
7
8           set.add("foo");
9           set.add("bar");
10          set.remove("foo");
11          System.out.println(set); // prints: [bar]
12      }
13  }
14
```

In the following examples `import java.util.*;` will be omitted.

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
```

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

Wrapper Class

Primitive data types can not be elements in collections. Use wrapper classes like *Integer* instead.

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

Generics

Generics

```
1         Object myStringAsObject = "klaus";
2         String myStringAsString = (String) myStringAsObject;
3
```

Generics

```
1         Object myStringAsObject = Integer.valueOf("42");
2         String myStringAsString = (String) myStringAsObject;
3
```

Why it won't work:

Integer can't be casted to String.

The Code before will compile but still cause an Exception in the JVM.

Generics

```
1        public class Box {
2            private Object object;
3
4            public void set(Object object) { this.object = object; }
5            public Object get() { return object; }
6        }
7
8
```

Generics

```
1          public class Box<T> {
2              // T stands for "Type"
3              private T t;
4
5              public void set(T t) { this.t = t; }
6              public T get() { return t; }
7          }
8
9          Box<Integer> integerBox; = new Box<Integer>();
10
11
```

Iterator

To iterate over a map use the iterator from the set of keys.

```
1     public static void main (String[] args) {
2
3         // [...] map, keys, values like previous slide
4         Iterator<Integer> iter = keys.iterator();
5
6         while(iter.hasNext()) {
7             System.out.print(map.get(iter.next()) + " ");
8         } // prints: bar foo bar
9
10        System.out.println(); // print a line break
11
12        for(Integer i: keys) {
13            System.out.print(map.get(i) + " ");
14        } // prints: bar foo bar
15    }
16
```

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)

```
1       Map<String, String> map = ...
2       for (Map.Entry<String, String> entry : map.entrySet()) {
3         System.out.println("Key: " + entry.getKey() +
4         ", value" + entry.getValue());
5       }
6
```