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

Generics

Generics

```java
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.

```java
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

```
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}

Box<Integer> integerBox; = new Box<Integer>();
```

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

Collections

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.

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`.

```java
import java.util.*;

public class TestSet {

    public static void main(String[] args) {
        Set<String> set = new HashSet<String>();

        set.add("foo");
        set.add("bar");
        set.remove("foo");
        System.out.println(set); // prints: [bar]
    }
}
```

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

List

A list is an ordered collection.

The implementation LinkedList is a double-linked list.

```java
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"

    System.out.println(list); // prints: [foo, bar, 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)

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

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

    list.add(1);
    list.add(3);
    list.add(3);
    list.add(7);

    for (Integer i : list) {
        System.out.print(i + " "); // prints: 1 3 3 7
    }
}
```

Iterator

An iterator iterates step by step over a collection.

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

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

    list.add(1);
    list.add(3);
    list.add(3);
    list.add(7);

    Iterator<Integer> iter = list.iterator();

    while (iter.hasNext()) {
        System.out.print(iter.next());
    }
    // prints: 1337
}
```

Iterator

A standard iterator has only three methods:

- boolean hasNext() - indicates if therer are more elements
- E next() - returns the next element
- void remove() - returns the current element

The iterator is instanced via collection.iterator() :

```
1       Collection <E> collection = new Implementation <E>;
2       Iterator <E> iter = collection.iterator ();
3
```

Special iterators like *ListIterator* are more sophisticated.

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.

```java
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]
}
```

Iterator

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

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

    // [...] map, keys, values like previous slide
    Iterator<Integer> iter = keys.iterator();

    while(iter.hasNext()) {
        System.out.print(map.get(iter.next()) + " ");
    } // prints: bar foo bar

    System.out.println(); // print a line break

    for(Integer i: keys) {
        System.out.print(map.get(i) + " ");
    } // prints: bar foo bar
}
```

Nested Maps

Nested maps offer storage with key pairs.

```java
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
```

19

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
```

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