Javakurs 6-13

Florian Pix 2018

Inhalt

- Generics – parametrisierte Klassen
- Collections

Generics

Parametrisierte Klassen

Generics - Parametrisierte Klassen

```
public class Box {
	private String string;

	Box(String string){
		this.string = string;
	}

	public void setString ( String string ) {
		this . string = string ;
	}

	public String getString () {
		return string ;
	}
}
```

Generics - Parametrisierte Klassen

```
public class genericBox<T> {
	private T t;

	genericBox(T t){
		this.t = t;
	}

	public T getT() {
		return t;
	}

	public void setT(T t) {
		this.t = t;
	}
}
```

Generics - Parametrisierte Klassen

```
public class Main {
    public static void main(String[] args) {
        Box normal = new Box("hallo");
^
        genericBox<String> stringBox = new genericBox<String>("hallo");

        genericBox<Integer> integerBox = new genericBox<Integer>(5);

        genericBox<Boolean> booleanBox = new genericBox<Boolean>(true);

    }
}
```

Collections

Interresante Effizienzvergleiche der verschiedenen Implementierungen

http://st.inf.tu-dresden.de/files/teaching/ss15/st/slides/21-st-collections.pdf

Collections

- Java bietet mit dem Collections framework verschiedenste Daten Strukturen an.
- Dazu zählen Sets, Listen, Maps und Queues.
- Das sind Interfaces zu denen es verschiedene Implementierungen mit verschiedenen Vorteilen gibt.
- Ihr könnt aber auch eure eigene Implementierung schreiben.

Set

- Ein Set ist eine Sammlung von Objekten eines Types.
- Es kann kein Element doppelt enthalten sein, bzw. jedes Element ist einzigartig
- Effektive Suche
- Keine Sortierung

Set

```
import java.util.HashSet;
import java.util.Set;

public class TestSet {
      public static void main ( String [] args ) {
            Set<String> set = new HashSet <String>() ;
            set.add (" foo ") ;
            set.add (" foo2 ") ;
            set.add (" bar ") ;
            set.add (" bar ") ;
            set.remove (" foo ");
            System . out . println ( set );
      }
}
```

List

- Sortierung / Reihenfolge
- Leichtes Durchgehen
- Keine effektive Suche

List

```
import java.util.LinkedList;
import java.util.List;

public class TestList {
      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 ]
      }
}
```

Wichtige Methoden für Listen

- 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

For Loop

- for (E e : collection) iteriert über jedes Element einer Sammlung

```
import java.util.LinkedList;
import java.util.List;

public class ForLoop {
	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

```
import java.util.*;

public class TestIterator {
        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

- boolean hasNext() sagt ob das Element einen Nachfolger hat
- E next() gibt den Nachfolger aus
- void remove() gibt das aktuelle Element zurück und "geht eins weiter"

- Instanziieren mittels collection.iterator()

Map

- Key – Value
- Keys sind einzigartig aber Werte können mehrfach auftauchen
- Sehr effektive Suche
- Durchgehen schwierig / langsam
- Kein Subinterface von Collections

Map

```
import java.util.HashMap;
import java.util.Map;

public class TestMap {
	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 }
	}
}
```

Nested Maps

```
import java.util.HashMap;
import java.util.Map;

public class TestNestedMap {
      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 ") ;
      }
}
```

Keyset und Valuecollection

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

Iterator für Map

- Wir können auch über Maps iterieren indem wir dessen Keyset nutzen.

- `Iterator < Integer > iter = keys . iterator () ;`

Lamdas for Maps

```
import java.util.HashMap;
import java.util.Map;

public class TestLambda {
	public static void main(String[] args) {
		Map<Integer, String> map = new HashMap();
		for(int i = 0; i < 10; i++) {
			map.put(i, "hallo");
		}

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

For Loop

```
import java.util.HashMap;
import java.util.Map;

public class TestForMap {
    public static void main(String[] args) {
        Map < String , String > map = new HashMap< String , String >();
        for ( Map . Entry < String , String > entry : map . entrySet () ) {
            System . out . println (" Key : " + entry . getKey () + ", value " + entry . getValue ());
        }
    }
}
```