

Objektorientierte Modellierung und Programmierung

Dr. Christian Schönberg

Java Development Kit (JDK)

Lernziele

- **JDK: List, Stack, Queue, Deque, Set, Map, Collection**
- **Iterable, Comparable**
- **Random, Math, Object, Class**

Übersicht über die Pakete des JDK

- **java.lang** Datentypen wie **Integer**, **Double**, **String** (kein **import** nötig)
- **java.util** Hilfsklassen, komplexe Datentypen
- **java.io** Input/Output → später
- **java.nio** nicht-blockierender Input/Output → später
- **java.awt** einfache GUI, Klassen wie **Color**, **Font**, **Image**
- **javax.swing** Swing GUI
- **javax.imageio** Input/Output für Bilder
- **java.lang.reflect** Reflection (Manipulation eines Programms zur Laufzeit)
- **java.net** Netzwerkzugriff
- **java.rmi** Remote Method Invocation
- **javax.xml** XML Unterstützung
- **java.sql** Datenbankzugriff
- **java.security** Sicherheit, Verschlüsselung
- **javax.crypto** Verschlüsselung
- ...

→ <https://docs.oracle.com/en/java/javase/13/docs/api/index.html>



Object
+ equals(obj: Object): boolean + hashCode(): int + toString(): String + getClass(): Class<?>


Drückt aus, dass zwei Objekte „gleich“ sind.

Wenn beide Objekte nicht **null** sind, repräsentiert **equals** eine Äquivalenzrelation:

- *reflexiv*: es gilt immer **x.equals(x)**
- *symmetrisch*: **x.equals(y) ⇔ y.equals(x)**
- *transitiv*: **x.equals(y) ∧ y.equals(z) ⇒ x.equals(z)**

Außerdem gelten:

- *konsistent*: **x.equals(y)** gilt immer (oder nie), solange **x** und **y** unverändert bleiben
- *eindeutiges null*: **x.equals(null)** gilt **nie**




Object
+ equals(obj: Object): boolean + hashCode(): int + toString(): String + getClass(): Class<?>

Gibt den Hash-Wert eines Objekts zurück. Wird verwendet von **HashMap** oder **HashSet**.
Es muss gelten:

- *konsistent*: **x.hashCode()** gibt immer den selben Wert zurück, solange **x** unverändert bleibt
- *respektiert equals*: wenn **x.equals(y)** gilt, muss auch **x.hashCode() == y.hashCode()** gelten
- *nicht eindeutig*: wenn **nicht x.equals(y)** gilt, darf trotzdem **x.hashCode() == y.hashCode()** gelten

Object



Object
+ equals(obj: Object): boolean + hashCode(): int + toString(): String + getClass(): Class<?>

Gibt eine String-Repräsentation des Objekts zurück.

System.out.println(Object) verwendet die **toString**-Methode zur Ausgabe.

Object

Object
+ equals(obj: Object): boolean + hashCode(): int + toString(): String + getClass(): Class<?>



Gibt die Laufzeit-Klasse des Objekts zurück.

Class

Class<T>
+ getCanonicalName(): String + getName(): String + getSimpleName(): String

Class

```
Object[] objects = { "a", new int[0], new int[0][0][0], new String[0] };

for (Object obj : objects) {
    Class<?> cls = obj.getClass();
    System.out.println(cls.getCanonicalName());
    System.out.println(cls.getName());
    System.out.println(cls.getSimpleName());
    System.out.println();
}
```

java.lang.String	int[][][]
java.lang.String	[[[I
String	int[][][]
int[]	java.lang.String[]
[I	[Ljava.lang.String;
int[]	String[]

```
Object strObj = "a";  
Object intObj = 1;
```

```
System.out.println(strObj.getClass() == String.class); // true  
System.out.println(intObj.getClass() == Integer.class); // true  
System.out.println(intObj.getClass() == Number.class); // false
```

```
System.out.println(strObj instanceof String); // true  
System.out.println(intObj instanceof Integer); // true  
System.out.println(intObj instanceof Number); // true
```

Beispiel: `equals`, `hashCode`, `toString`

```
public class Domino {  
  
    // ...  
  
    @Override  
    public String toString() {  
        return "Domino piece with " + dots + " dots";  
    }  
  
    @Override  
    public int hashCode() {  
        return dots;  
    }  
}
```

Beispiel: equals, hashCode, toString

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Domino other = (Domino) obj;
    if (dots != other.dots)
        return false;
    return true;
}
}
```

Beispiel: equals, hashCode, toString

```
Domino d1 = new Domino(3);
Domino d2 = new Domino(3);
Domino d3 = new Domino(4);

System.out.println(d1.hashCode()); // 3
System.out.println(d2.hashCode()); // 3
System.out.println(d3.hashCode()); // 4

System.out.println(d1.equals(d2)); // true
System.out.println(d1.equals(d3)); // false

Integer i = Integer.valueOf(3);
System.out.println(i.hashCode()); // 3
System.out.println(d1.equals(i)); // false

Set<Domino> set = new HashSet<>();
set.add(d1); set.add(d2); set.add(d3);
System.out.println(set);
// [Domino piece with 3 dots, Domino piece with 4 dots]
```

Math	
- E: double {readOnly}	
- PI: double {readOnly}	
+ abs(v: double): double	
+ ceil(v: double): double	}
+ floor(v: double): double	
+ round(v: double): long	}
+ cos(v: double): double	
+ sin(v: double): double	}
+ tan(v: double): double	
+ pow(a: double, b: double): double	}
+ log(v: double): double	
+ sqrt(v: double): double	
+ random(v: double): double	}
+ min(a: double, b: double): double	
+ max(a: double, b: double): double	

Konstanten e (Basis des natürlichen Logarithmus) und π

|v| (Absolutbetrag)

Auf- und Abrunden zur nächsten natürlichen Zahl

Trigonometrische Funktionen

Potenz, Logarithmus (Basis e, auch: **log10**), Wurzel

Zufallszahl zwischen 0 (einschließlich) und 1 (ausschließlich)

Minimum und Maximum

Random
+ Random() + Random(seed: long) + nextBoolean(): boolean + nextDouble(): double + nextFloat(): float + nextInt(): int + nextInt(bound: int): int + nextLong(): long + nextGaussian(): double

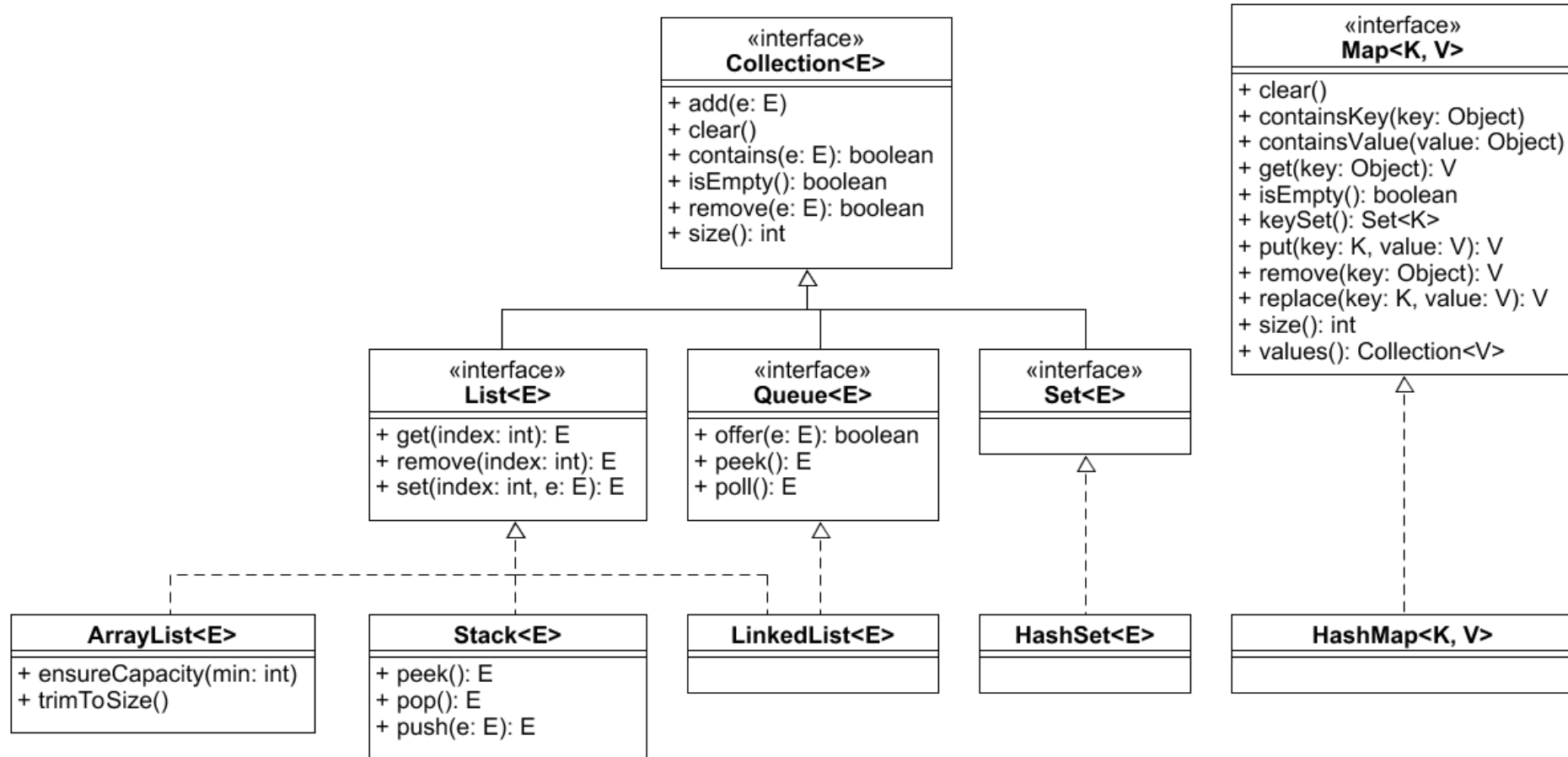
seed: legt den Initialwert des Pseudo-Zufallszahlen-Generators fest

der zurückgegebene Wert v ist $0 \leq v < 1$

bound: der zurückgegebene Wert v ist $0 \leq v < \mathbf{bound}$

der zurückgegebene Wert folgt der Normalverteilung mit dem Durchschnitt 0 und der Standardabweichung 1

Java Collections-API



```
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.HashSet;
```

```
Collection<Integer> ints = new ArrayList<>();  
ints.add(1);  
ints.add(1);  
ints.add(2);  
// [1, 1, 2]
```

```
ints = new HashSet<>();  
ints.add(1);  
ints.add(1);  
ints.add(2);  
// [1, 2]
```

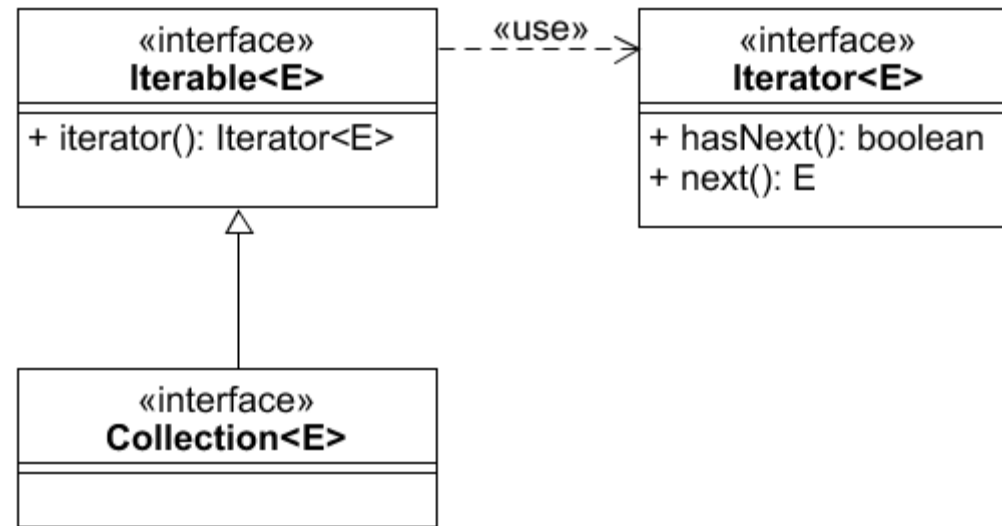
```
import java.util.HashMap;  
import java.util.Map;
```

```
Map<String, Integer> map = new HashMap<>();  
map.put("one", 1);  
map.put("two", 2);  
// "one" -> 1  
// "two" -> 2  
System.out.println(map.get("one")); // 1
```

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

```
Map<String, List<Integer>> intMap = new HashMap<>();  
List<Integer> ones = new ArrayList<>();  
ones.add(1);  
List<Integer> twos = new ArrayList<>();  
twos.add(2);  
twos.add(2);  
intMap.put("ones", ones);  
intMap.put("twos", twos);  
// "ones" -> [1]  
// "twos" -> [2, 2]  
System.out.println(intMap.get("twos").get(1)); // 2
```

Iterator



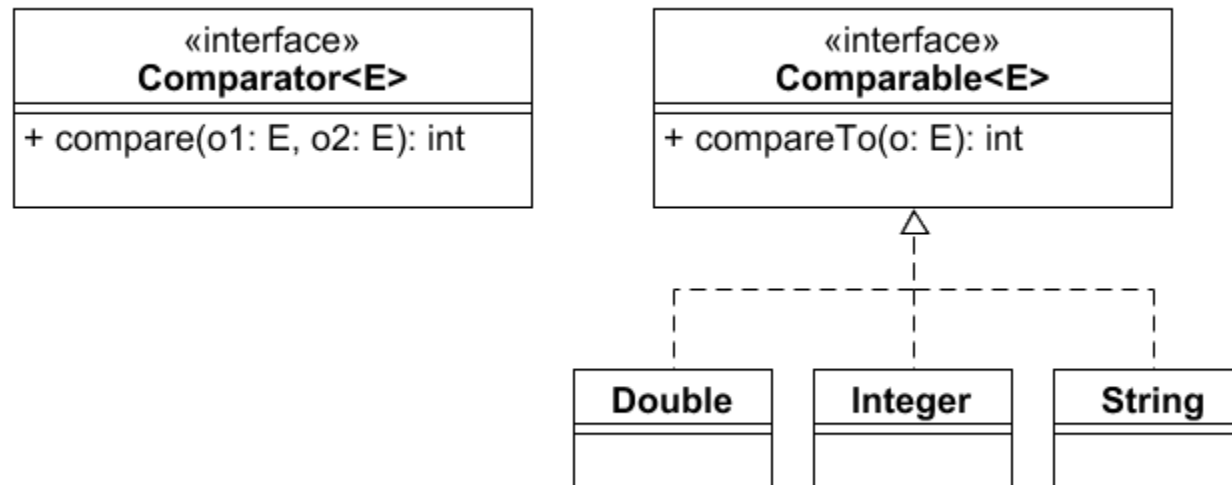
Beispiel: **Iterator**

```
List<Integer> intList = new ArrayList<>();
Random random = new Random();
for (int i = 0; i < 10; i++) {
    intList.add(random.nextInt(100));
}

Iterator<Integer> iterator = intList.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}

for (int i : intList) {
    System.out.println(i);
}
```

Comparator, Comparable



`compare(o1, o2) == o1.compareTo(o2)`

- `< 0`, falls **`o1 < o2`**
- `= 0`, falls **`o1 = o2`**
- `> 0`, falls **`o1 > o2`**

Beispiel: Comparable

```
List<Integer> intList = new ArrayList<>();  
intList.add(3);  
intList.add(1);  
intList.add(2);  
// use the Comparable interface to determine the order of the elements  
intList.sort(null);  
// [1, 2, 3]
```

Beispiel: Comparator

```
List<Object[]> listOfArrays = new ArrayList<>();  
listOfArrays.add(new Integer[] { 1, 2, 3 });  
listOfArrays.add(new String[] { "a", "b" });  
listOfArrays.add(new Double[] { 4.0 });  
// use the custom Comparator to determine the order of the elements  
listOfArrays.sort(new ArrayComparator());
```

```
class ArrayComparator implements Comparator<Object[]> {  
    @Override  
    public int compare(Object[] o1, Object[] o2) {  
        if (o1 == null && o2 == null) {  
            return 0;  
        }  
        if (o1 == null) {  
            return -1;  
        }  
        if (o2 == null) {  
            return 1;  
        }  
        return o1.length - o2.length;  
    }  
}
```

```
// [ { 4.0 }, { "a", "b" }, { 1, 2, 3 } ]
```

Werkzeug-Klasse **Collections**

```
package java.util;

import java.util.Comparator;
import java.util.List;

public class Collections {

    public static <T extends Comparable<T>>
        void sort(List<T> list) { }

    public static <T>
        void sort(List<T> list, Comparator<T> comp) { }

}
```

* leicht vereinfachte Darstellung

Werkzeug-Klasse **Collections**

```
package java.util;

import java.util.Comparator;
import java.util.List;

public class Collections {

    public static <T extends Comparable<? super T>>
        void sort(List<T> list) { }

    public static <T>
        void sort(List<T> list,
                  Comparator<? super T> comp) { }

}
```

* ausführliche Darstellung

Beispiel: Comparable (2)

```
List<Integer> intList = new ArrayList<>();  
intList.add(3);  
intList.add(2);  
intList.add(5);  
intList.add(1);  
intList.add(4);  
Collections.sort(intList);  
// [1, 2, 3, 4, 5]
```

Beispiel: Comparator (2)

```
class EvenBeforeOdd implements Comparator<Integer> {  
  
    @Override  
    public int compare(Integer o1, Integer o2) {  
        if (o1 % 2 == 0) {  
            if (o2 % 2 == 0) {  
                return o1 - o2;  
            } else {  
                return -1;  
            }  
        } else {  
            if (o2 % 2 == 0) {  
                return 1;  
            } else {  
                return o1 - o2;  
            }  
        }  
    }  
}
```

Beispiel: Comparator

```
Collections.sort(intList, new EvenBeforeOdd());  
// [2, 4, 1, 3, 5]
```

Lernziele

- **JDK: List, Stack, Queue, Deque, Set, Map, Collection**
- **Iterable, Comparable**
- **Random, Math, Object, Class**