

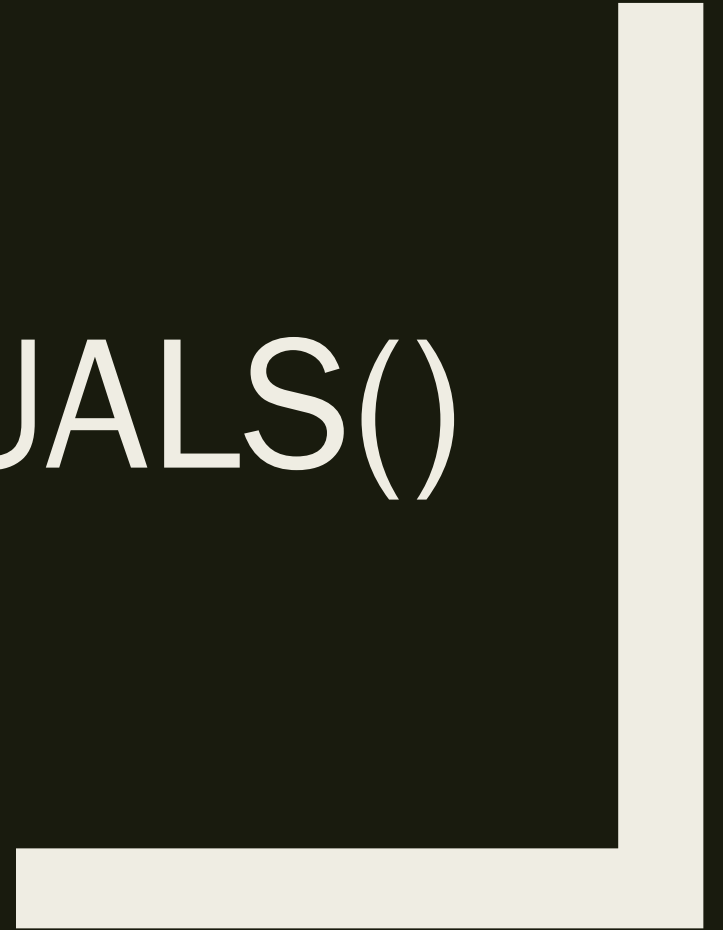


EQUALS UND HASHCODE

Die Methoden equals() und hashCode()

- Objectgleichheit mit equals()
- Objectidentifizierung mit hashCode()
- Arbeitsweise eines HashSets
 - *Klasse Person, HashSet<Person> personen*
 - Erweiterung der Klasse Person<Person> mit den Methoden equals() und hashCode()
- Der hashCode() contract – OCP Exam Guide Seite 344
- Der equals() contract – OCP Exam Guide Seite 339

EQUALS()



Objektgleichheit mit equals()

- Die Basisklasse Object enthält eine Standardimplementierung der equals() Methode

```
public boolean equals(Object obj) {  
    return (this == obj);  
}
```

- Die Standardimplementierung testet die Identität.
- Es wird also geprüft ob zwei Referenzen dasselbe Objekt repräsentieren.

Objektgleichheit mit equals()

- equals() sollte Objekte auf Gleichheit prüfen
 - *Dafür muss die Standardimplementierung überschrieben werden.*
- Was bedeutet die Gleichheit zweier Objekte?
 - *Der Vergleich zweier Objekte sollte gewissen Regeln folgen*
 - *Diese sind formal im equals() contract beschrieben*
 - [https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals\(java.lang.Object\)](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals(java.lang.Object))

Objektgleichheit mit equals()

■ Aus der apidoc

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

Objektgleichheit mit equals()

- Originalbeschreibung des equals() contracts

Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes.

- Die Überschreibung der equals Methode erfordert die korrekte Überschreibung der hashCode() Methode

HASHCODE()

Objektidentifizierung mit hashCode()

- Die Basisklasse Object enthält eine native Standardimplementierung der hashCode() Methode

```
public native int hashCode();
```

- Zur api doc
 - [*https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode\(\)*](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode())

Objektidentifizierung mit hashCode()

- Beschreibung der hashCode() Methode in dem JavaDoc der Klasse Object:

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)

- Die Default-Hash-Code-Berechnung basiert also auf der Adresse des Objekts
 - *Genaueres nicht bekannt*

Objektidentifizierung mit hashCode()

- Die Methode hashCode() soll zu jedem Objekt eine möglichst eindeutige Integerzahl (sowohl positiv als auch negativ) liefern, die das Objekt identifiziert.
 - *Die Ganzzahl heißt Hash-Code bzw. Hash-Wert*
 - *Hash-Codes werden mit Hash-Funktionen berechnet.*
 - *hashCode() ist die Implementierung einer Hash-Funktion*
- Hash-Funktionen sind mathematische Funktionen, die sich schnell berechnen lassen und einen breit gestreuten Wertebereich haben
 - *Eine kleine Änderung der Ausgangsdaten führt oft zu einem völlig anderen Hash-Code*
 - *Damit können die Objekte möglichst gut verteilt werden. Ein bestimmtes Objekt erhält stets denselben Hash-Code*

Objektidentifizierung mit hashCode()

- Nötig sind Hashcodes, wenn Objekte in Datenstrukturen untergebracht werden, die nach dem Hashing-Verfahren arbeiten.
- Datenstrukturen, die nach dem Hashing-Verfahren arbeiten:
 - *HashSet<E>*
 - *LinkedHashSet<E>*
 - *HashMap<K,V>*
 - *HashTable<K,V>*

Objektidentifizierung mit hashCode()

- Was genau ist die Anforderung eine Implementierung von hashCode, die konsistent zu equals() ist?
 - Die Anforderungen an hashCode im sogenannten hashCode-Contract beschrieben.
- Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
 - If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
 - It is *not* required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.

ARBEITSWEISE EINES HASHSETS()

Arbeitsweise eines HashSets

- Die Klasse `HashSet<E>` Verwendet zur Speicherung von Objektreferenzen das sogenannte Hashing(auch Streuspeicher-Verfahren genannt)
- Beim Hashing wird jedem Objekt eine Zahl, der so genannte `HashCode`, zugeordnet
- Der Hash-Code dient als Index in einer Tabelle, in der Objekte gespeichert werden
- Um zu prüfen ob ein Objekt in der Tabelle enthalten ist, wird der Tabellenindex jedes Mal erneut aus dem Hash-code berechnet
 - *Der Tabellenindex ermöglicht den direkten Zugriff*
 - *Durch den direkten Tabellenzugriff lässt sich ein Objekt sehr viel schneller wieder finden als durch sequenzielles suchen*

Arbeitsweise eines HashSets

- In der Praxis ist die Tabelle in der Regel kleiner als der Wertebereich der Hash-Funktion
 - *Codes werden daher mithilfe des Restwertoperator (%)s auf die tatsächliche Tabellengröße abgebildet*
- Es kann vorkommen dass zwei verschiedene Objekte den gleichen Hash-Code haben bzw. dass ihr Hash-Code durch die Berechnung auf den gleichen Index abgelegt werden
- In der Tabelle werden nicht einzelne Objekte sondern Listen(sog. Eimer, engl. Buckets) von Objekten mit den gleichen Code gespeichert.
 - *Die Listen sind in der Regel sehr klein und daher schnell zu durchsuchen*

Arbeitsweise eines HashSets

- Methode hashCode() berechnet den Hash - Code des Objekts
 - *Der Hash-Code wird mit dem Restwert-Operator auf die Tabellengröße angepasst, um die richtige Liste zu finden*
- Mit Hilfe der equals()-Methode wird in der Liste nach dem richtigen Objekt gesucht
 - *Das richtige Objekt muss nicht das ursprünglich eingefügte Objekt sein, da die equals()-Methode lediglich auf inhaltliche Gleichheit prüft*

Einführendes Beispiel


- Ausgangsklasse Person

```
public class Person{

    private String vorname;
    private String nachname;
    private int alter;
    private String anschrift;

    public Person(String vorname, String name,
                  int alter, String anschrift) {
        this.vorname = vorname;
        this.nachname = name;
        this.alter = alter;
        this.anschrift = anschrift;
    }
}
```

<<Java Class>>

 Person

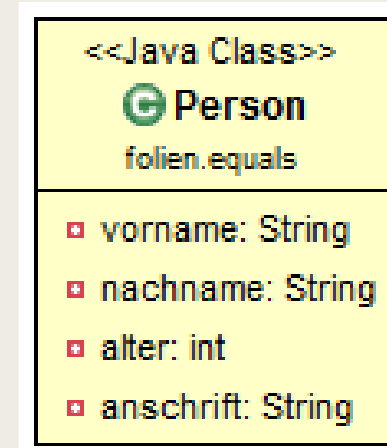
folien.equals

- ✚ vorname: String
- ✚ nachname: String
- ✚ alter: int
- ✚ anschrift: String

Einführendes Beispiel

- Ausgangsklasse Person

```
@Override
public String toString() {
    return "\n" + vorname + " " + nachname + " "
+ alter + " wohnhaft in: " + anschrift;
}
```



Einführendes Beispiel

- 3 inhaltlich ungleiche Person Objekte werden erzeugt

```
Person martinJun =  
    new Person("Martin", "Mueller", 25,  
        "0000 Musterstadt, Musterstrasse 0");  
  
Person angelika = new Person("Angelika", "Mueller", 25,  
    "0000 Musterstadt, Musterstrasse 0");  
  
Person martinSen =  
    new Person("Martin", "Mueller", 60,  
        "0000 Musterstadt, Musterstrasse 0");
```

Einführendes Beispiel

- Die 3 unterschiedlichen Objekte der Klasse Person werden in das HashSet<Person> personen eingefügt

```
Set<Person> personen = new HashSet<Person>();  
  
personen.add(angelika);  
personen.add(martinJun);  
personen.add(martinSen);  
  
System.out.println(personen);
```

- HashSet<Person> personen enthält 3 Objekte:

```
Martin Mueller 60 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Angelika Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Martin Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0]
```

Einführendes Beispiel

- 4 Person Objekte werden erzeugt (2 inhaltlich gleich)

```
Person martinJun =  
    new Person("Martin", "Mueller", 25,  
        "0000 Musterstadt, Musterstrasse 0");  
  
Person angelika = new Person("Angelika", "Mueller", 25,  
    "0000 Musterstadt, Musterstrasse 0");  
  
Person martinSen =  
    new Person("Martin", "Mueller", 60,  
        "0000 Musterstadt, Musterstrasse 0");  
  
Person angelika2 = new Person("Angelika", "Mueller", 25,  
    "0000 Musterstadt, Musterstrasse 0");
```

Einführendes Beispiel

- Die 4 Personen Objekte werden in das HashSet<Person> personen eingefügt

```
Set<Person> personen = new HashSet<Person>();  
  
personen.add(angelika);  
personen.add(martinJun);  
personen.add(martinSen);  
personen.add(angelika2);  
  
System.out.println(personen);
```

- HashSet<Person> personen enthält 4 Objekte

```
Martin Mueller 60 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Angelika Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Martin Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Angelika Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0]
```

Einführendes Beispiel

- HashSet<Person> personen enthält 4 Objekte

```
Martin Mueller 60 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Angelika Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Martin Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Angelika Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0]
```

- Ein Set – hier HashSet - ist eine im mathematischen Sinne definierte Menge von Objekten.
 - *Ein Set darf keine doppelten Elemente enthalten*
- HashSet<Person> personen enthält zwei inhaltlich gleiche Objekte
 - *Ein Element ist doppelt enthalten*

Einführendes Beispiel

- Weshalb enthält nun das `HashSet<Person>` personen 4 Objekte, insbesondere zwei inhaltlich Gleiche?
- Dies hat mit der Arbeitsweise eines HashSets zu tun
 - *Zunächst wird der Hash-Code des jeweiligen Objekts berechnet, dieser dient dann als Tabellenindex*
 - *Die Klasse Person überschreibt die `hashCode()` Methode nicht*
 - *Die Berechnung des Hash-Codes erfolgt mit der nativen `hashCode()` Methode, diese basiert auf der Adresse des jeweiligen Objekts*
 - *Da die vier Objekte unterschiedliche Speicheradressen haben, ergeben sich vier Hash-Codes*
 - *Unterschiedliche Hash-Codes bedeuten unterschiedliche Objekte, gemäß des Hash-Code Contracts*
 - bei korrekter Implementierung der Methoden `equals()` und `hashCode()`

Einführendes Beispiel

- Überschreiben von equals() in der Klasse Person

```
@Override
public boolean equals(Object o) {
    if (o instanceof Person) {
        Person other = (Person) o;
        return (this.vorname.equals(other.vorname) &&
                this.nachname.equals(other.nachname) &&
                this.alter== other.alter &&
                this.anschrift.equals(other.anschrift));
    }
    return false;
}
```

Einführendes Beispiel

- Überschreiben von hashCode() in der Klasse Person

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + alter;
    result = prime * result + ((anschrift == null) ?
        0 : anschrift.hashCode());
    result = prime * result + ((nachname == null) ?
        0 : nachname.hashCode());
    result = prime * result + ((vorname == null) ?
        0 : vorname.hashCode());
    return result;
}
```

Einführendes Beispiel

- Die 4 Personen Objekte werden in das HashSet<Person> personen eingefügt

```
Set<Person> personen = new HashSet<Person>();  
  
personen.add(angelika);  
personen.add(martinJun);  
personen.add(martinSen);  
personen.add(angelika2);  
  
System.out.println(personen);
```

- HashSet<Person> personen hat 3 Personen Objekte

```
Angelika Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Martin Mueller 25 wohnhaft in: 0000 Musterstadt, Musterstrasse 0,  
Martin Mueller 60 wohnhaft in: 0000 Musterstadt, Musterstrasse 0]
```

Einführendes Beispiel

- Nach Überschreiben der Methoden equals() und hashCode() sind die Ergebnisse korrekt
 - *HashSet<Person> personen* enthält 3 Personen Objekte
 - Von den zwei inhaltlich gleichen Objekten wurde wie erwartet, lediglich ein Objekt in das *HashSet<Person> personen* eingefügt
 - Die zwei gleichen Objekte haben nun gleiche Hash-Codes durch das Überschreiben der hashCode() Methode
 - Es wird folglich nur ein Objekt – von den zwei gleichen – in das *HashSet<Person> personen* eingefügt

Literaturverzeichnis

- Hash-Code-Berechnung: Wie, wann und warum implementiert man die hashCode()-Methode? JavaSPEKTRUM, Mai 2002 von Klaus Kreft & Angelika Langer
 - <http://www.angelikalanger.com/Articles/EffectiveJava/03.HashCode/03.HashCode.html>
- Objektvergleich: Wie, wann und warum implementiert man die equals()-Methode? Teil 1: Die Prinzipien der Implementierung von equals(); JavaSPEKTRUM, Januar 2002 von Klaus Kreft & Angelika Langer
 - <http://www.angelikalanger.com/Articles/EffectiveJava/01.Equals-Part1/01.Equals1.html>
- Java ist auch eine Insel: Einführung, Ausbildung, Praxis; Rheinwerk Computing; von Christian Ullenboom
 - <http://openbook.rheinwerk-verlag.de/javainsel/>