# Objektorientierte Programmierung

Die Klasse Object und
die Methoden equals() und hashCode()

Prof. Dr. Ulrike Hammerschall
Fakultät für Informatik und Mathematik

## Die Basisklasse Object

- Die Klasse *java.lang.Object* ist Basisklasse aller Klassen in Java.
- Folgende Definitionen sind daher äquivalent:

```
public class MyClass {...}
```

- und

```
public class MyClass extends Object {...}
```

- Konsequenz:
  - Alle Java-Klassen haben - direkt oder indirekt - Object als gemeinsame Basisklasse.
  - Alle Java-Klassen erben die Methoden der Klasse Object.

## Vordefinierte Methoden der Klasse Object

```
// Eindeutige Textdarstellung des Objekts (Default: Name + ObjektID)
public String toString()
// Wertevergleich von Objekten
public boolean equals(Object x)
// Eindeutiger Hashwert für das Objekt
public int hashCode()
// Erzeugt flache Kopie des Objekts
protected Object clone()
// Liefert Typ des Objekts
public Class getClass()
```



## Listen und die Methode equals()

- Listen arbeiten zum Vergleich von Elementen mit der Methode *equals(…)*: z.B. *indexOf*, *lastIndexOf*, *remove*, *contains*, …
- Generell erbt jede Klasse die Implementierung der equals-Methode der Klasse *Object*. Diese prüft ausschließlich auf Identität von Objekten.
- Um Objekte auf Wertegleichheit zu prüfen ist daher eine Redefinition der equals-Methode in den jeweiligen Klassen notwendig.
- Eine fehlende Redefinition der equals-Methode kann daher unbemerkt zu fehlerhaftem Verhalten von Listen führen.
- Für alle Elemente, die in einer Liste verwaltet werden, ist daher Redefinition der equals-Methode sinnvoll.

## Die Methode toString()

HM

- Liefert eine String-Repräsentation des Objekts zusammengesetzt aus Klassennamen und Hash-Wert.
- Implementierung der Methode in Object:

```
public String toString() {
       return getClass().getName() + "@" +
       Integer.toHexString(hashCode());
}
```

- Liefert bei unterschiedlichen Objekten unterschiedliche Werte.
- Die Methode toString sollte bei Klassen, die einen Zustand haben, in der Regel überschrieben werden.



## Regeln zur Implementierung der Methode equals()

HM

- Reflexivität: Für jedes Objekt x gilt

  `x.equals(x) → true`

- Symmetrie: Für zwei Objekte x und y gilt

  `x.equals(y) == y.equals(x)`

- Transitivität: Wenn für drei Objekte x, y und z

  `x.equals(y) → true und`

  `y.equals(z) → true gilt, dann auch`

  `x.equals(z) → true`

- null-Vergleich: Für alle Objekte x gilt

  `x.equals(null) → false`

vgl: https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/Object.html#equals(java.lang.Object)

## Beispiel mit Vererbung

*Basisklasse Person*

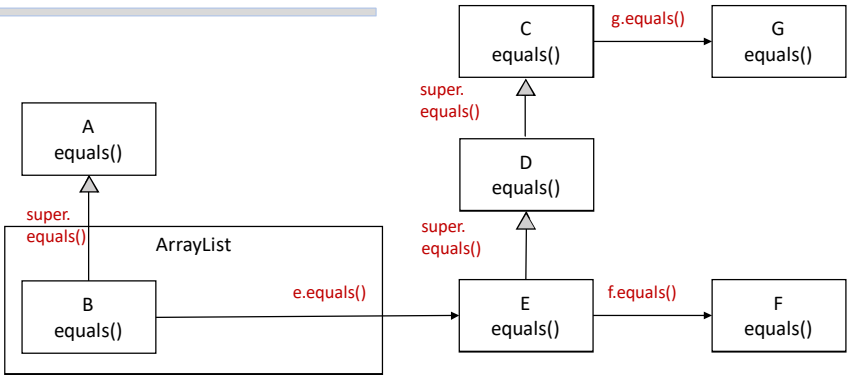
```
@Override
public boolean equals(Object obj) {
        if (this == obj)
                return true;
        if (obj == null)
                return false;
        if (getClass() != obj.getClass())
                return false;
        Person other = (Person) obj;
        if (id != other.id)
    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

*Abgeleitete Klasse Student*

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



## Equals bei Vererbung und Assoziationen




Für alle Elementtypen in Container-Klassen muss die korrekte Implementierung der equals Methode in allen durch Vererbung oder Assoziationen betroffenen Klassen sichergestellt sein.
Relevant sind alle Klassen die selbst Objektvariablen halten und somit einen Zustand repräsentieren.

## Collections und die Methode hashCode

HM

- Die Methode *hashCode()* wird bei einigen Collection-Klassen ergänzend zu *equals()* verwendet um die Effizienz bei der Elementsuche zu steigern.
- Ein Hashcode ist ein für ein gegebenes Objekt charakteristischer int-Wert.
- Die Methode *hashCode()* berechnet den Hashwert eines Objekts z.B. anhand der aktuellen Werte der Objektvariablen. Der konkrete Zahlenwert ist dabei irrelevant.
- Die Methode *hashCode()* wird insbesondere von *HashSet* (und *HashMap*) zur effizienten Suche verwendet.
- Sie wird immer gemeinsam mit der Methode *equals()* redefiniert.



## Redefinition der Methode hashCode() in der Klasse String

HM

```
public int hashCode() {
    int h = hash;
    if (h == 0 && !hashIsZero) {
        h = isLatin1() ? StringLatin1.hashCode(value) : StringUTF16.hashCode(value);
        if (h == 0) {
            hashIsZero = true;
        } else {
            hash = h;
        }
    }
    return h;
}
```

## Eigenschaften des Hash-Algorithmus

HM

- Geringe Wahrscheinlichkeit für Kollisionen (keine gleichen Hashwerte für unterschiedliche Objekte).
- Geringer Speicherbedarf des Hashcodes.
- Ähnliche Eingabewerte sollten zu möglichst unterschiedlichen Hashcodes führen.
- Breite Streuung der Ergebnisse: Jeder Wert im Wertebereich sollte möglichst erreicht werden.
- Effiziente Berechnung des Algorithmus.
- Ordnungserhaltend bei vergleichendem Zugriff



## Zusammenhang hashCode() und equals()

HM

- Abhängigkeit zwischen equals und hashCode

| | | |
|---|---|---|
| x.equals(y) => true | ⇒ | Es muss gelten x.hashCode() == y.hashCode() |
| x.equals(y) => false | ⇒ | Es sollte gelten x.hashCode() != y.hashCode() |

- Erste Anforderung Pflicht
- Zweite Anforderung optional ⇒ Einfluss auf Effizienz
- Vorgehen:
  - Die Methode hashCode dient zur schnellen Suche ähnlicher Werte.
  - Die Methode equals führt im Anschluss den tatsächlichen Vergleich durch.

- *Gemeinsam handelt es sich um einen effizienten Mechanismus zur Suche von Objekten in HashSets und HashMaps -> Thema nach den Weihnachtsferien*