Inhaltsverzeichnis

1. Der Objektvertrag

In Java **erben** alle Klassen automatisch von der Klasse

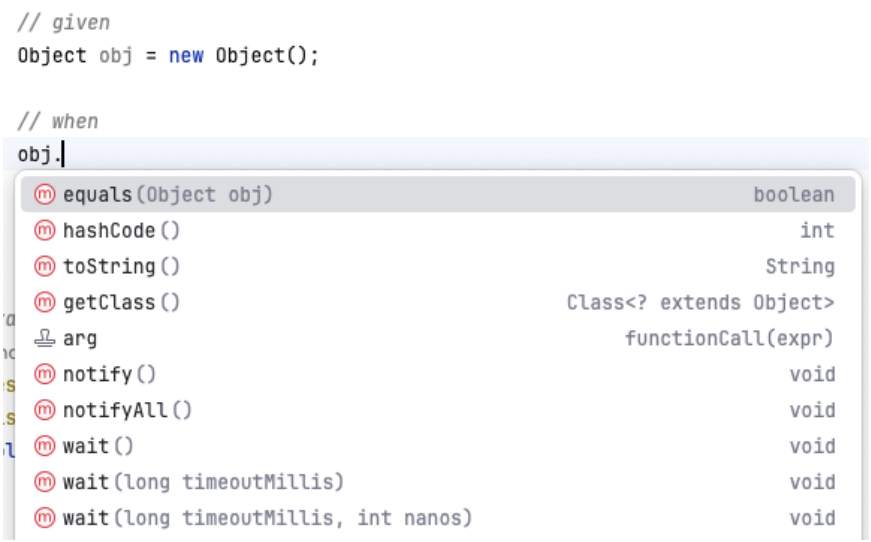
```
java.lang.Object
```

denn *alle Klassen* in Java *sind* "Objekte", daher wird Java selbst auch als

```
objektorientierte Programmiersprache
```

bezeichnet. Die Definition dieser obersten Klasse ist natürlich dokumentiert: → https://docs.oracle.com/.../java/lang/Object.html

Jedes Objekt in Java hat eine **grundlegende Menge an Funktionalitäten**, wie im Bild dargestellt:



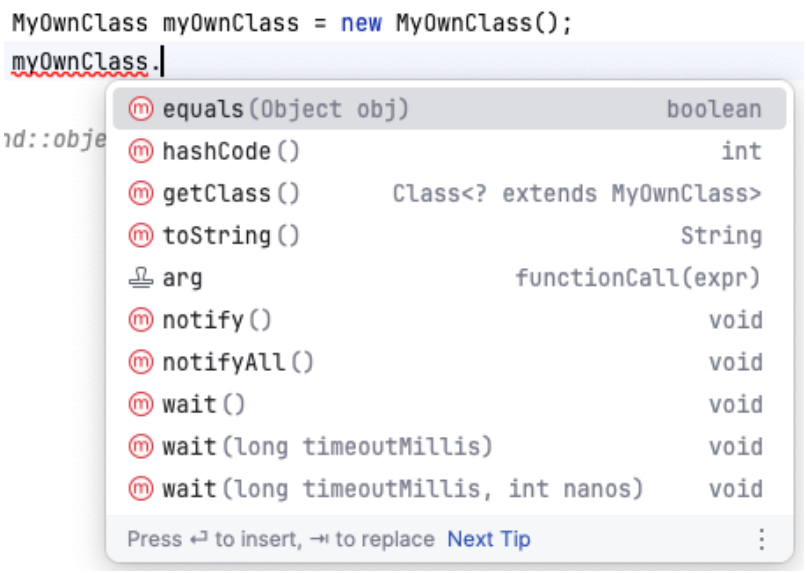
Alle Klassen in Java **erben** von der Klasse `Object`:

```
public class MyOwnClass extends Object {}
```

Da dies zu 100% für *alles* und *immer* gilt, muss es nicht explizit hingeschrieben werden, sodass folgendes z.B. beim Implementieren einer neuen Klasse vollkommen ausreicht:

```
public class MyOwnClass {} // ohne 'extends'
```

Obwohl weder direkt sichtbar noch explizit implementiert, sind die **Methoden**, die von `Object` bereitgestellt werden, durch die **Vererbung** dennoch nutzbar:



Die Klasse stellt also eine Art **Grundvertrag** (*Object Contract*) für alle Java Klassen dar und bietet eine grundlegende Menge an Funktionalitäten für alle Java Klassen.

Von diesen **Methoden** sollen die folgenden drei etwas näher betrachtet werden, da sie eine besondere Bedeutung haben:

- `equals()`
- `hashCode()`
- `toString()`

Die `Object`-Klasse definiert - neben anderen - diese Methoden. Sie werden im Folgenden detailliert erläutert. Die `toString()` Methode ist vor allem deshalb interessant, wie sie die Möglichkeit bietet, eine String-Repräsentation des jeweiligen Objektes zu implementieren (dazu später mehr).

1.1. Objektidentität mit `hashCode()`

Instanzen von Klassen - also Objekte - besitzen eine implizite **Identität**, eine Zahl.

*Was bedeutet diese **Objektidentität**?*

Einfach ausgedrückt, gibt `hashCode()` einen **ganzzahligen Wert** zurück, der von einem *Hashing-Algorithmus* generiert wird. Anhand dieser Zahl wird standardmäßig entscheiden, ob Instanzen *identisch* sind oder nicht.

Beim Vergleich von Objekten/Instanzen ist aber die *Gleichheit* von der *Identität* zu unterscheiden!

Etwas "ungenau" formuliert kann man sagen:

1. **Identität** zweier Instanzen bedeutet i.d.R. *technisch/referenziell* gleich
2. **Gleichheit** zweier Instanzen bedeutet i.d.R. *fachlich* gleich

Der allgemeine **Vertrag** von `hashCode()` besagt:

- Immer wenn `hashCode()` während der Ausführung einer Java-Anwendung mehr als einmal für dasselbe Objekt aufgerufen wird, muss es konsistent **denselben** Wert zurückgeben. Vorausgesetzt, dass keine Informationen geändert werden, die in Gleichheitsvergleichen für das Objekt verwendet werden. Dieser Wert muss nicht von einer Ausführung einer Anwendung zur anderen Ausführung derselben Anwendung konsistent bleiben.
- Wenn zwei Objekte gemäß der Methode `equals(Object)` **gleich** sind, muss der Aufruf der Methode `hashCode()` für jedes der beiden Objekte denselben Wert erzeugen.
- Wenn zwei Objekte gemäß der Methode `equals(Object)` **ungleich** sind, muss der Aufruf der Methode `hashCode()` für jedes der beiden Objekte nicht zwingend zu unterschiedlichen ganzzahligen Ergebnissen führen. Man sollte sich jedoch darüber im Klaren sein, dass die Leistung von Hash-Tabellen durch die Erzeugung unterschiedlicher ganzzahliger Ergebnisse für ungleiche Objekte verbessert wird.
- „Soweit es einigermaßen praktikabel ist, gibt die von der Klasse `Object` definierte Methode `hashCode()` eindeutige Ganzzahlen für verschiedene Objekte zurück.“

Viele *interne* Methoden und/oder Prozesse, die von der Java API angeboten werden, nutzen im Hintergrund die hier diskutierten Prinzipien, z.B. bei `HashMaps`, bei der der Begriff `hash` schon im Namen verankert ist.

1.2. Objektgleichheit mit `equals()`

Die Standardimplementierung von `equals()`, d.h. wenn sie "nur" von Object geerbt wird und *nicht* in der (neuen) Klasse implementiert wird, besagt, dass **Objekt-Gleichheit** dasselbe ist wie **Objekt-Identität**.

Das lässt sich besser am Code erläutern:

Gleich oder Identisch?

```
@Test
@DisplayName("Demo 1: Instances Not Equal")
public void cannotCommitEqualInstances() {
    // given
    Passenger passenger1 = new Passenger("Brad Pitt");
    Passenger passenger2 = new Passenger("Brad Pitt");

    // when
    boolean areEqual = passenger1.equals(passenger2);

    // then
    assertFalse(areEqual);
    System.out.printf(
            "%s != %s  %n",
```

```
            passenger1.hashCode(),
            passenger2.hashCode());
}
```

Gleich oder Identisch?

```
@Test
@DisplayName("Demo 2: Instance Equality")
public void canCommitEqualInstances() {
    // given
    Train train1 = new Train("RB-10");
    Train train2 = new Train("RB-10");

    // when
    boolean areEqual = train1.equals(train2);

    // then
    assertTrue(areEqual);
    System.out.printf(
            "%s == %s  %n",
            train1.hashCode(),
            train2.hashCode());
}
```

1.3. Objektrepräsentation mit `toString()`

Die Standardimplementierung von `toString` ist eine Kombination von **Klassenname** und **HashCode** (Identität):

```
getClass().getName() + "@" + Integer.toHexString(hashCode());
```

Die Zahl rechts vom `@` Symbol ist der HashCode, nur konvertiert in einen vorzeichenlosen, ganzzahligen (Integer-) Wert im *Hexadezimalformat* (Basis 16).

Als **Beispiel**, der Aufruf von

```
Object obj = new Object();
String str = obj.toString();
```

ergibt eine Zeichenkette `str` ähnlich dem Folgenden:

```
java.lang.Object@76b10754
```