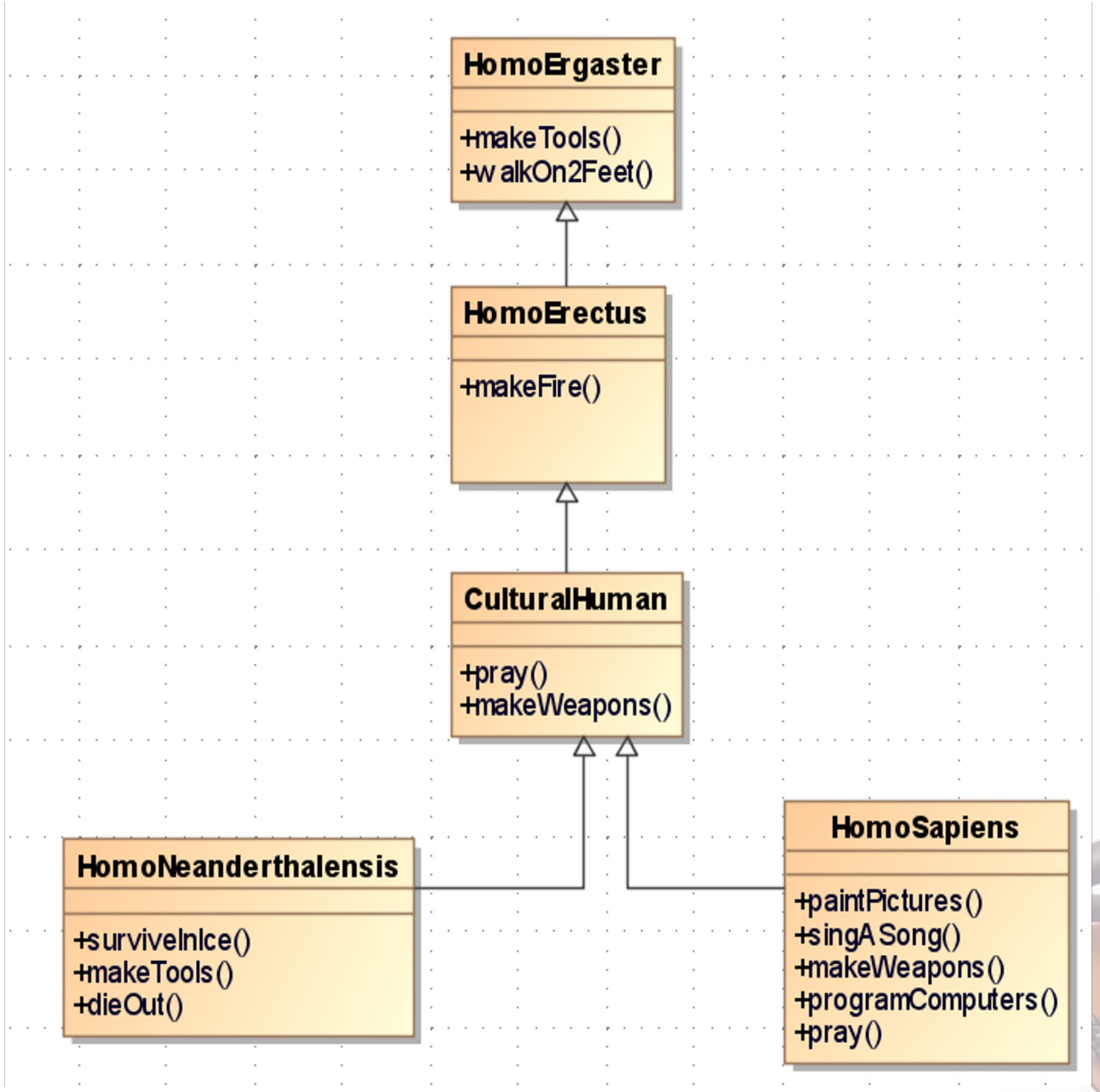
Java Fundamentals

Inheritance (Vererbung)

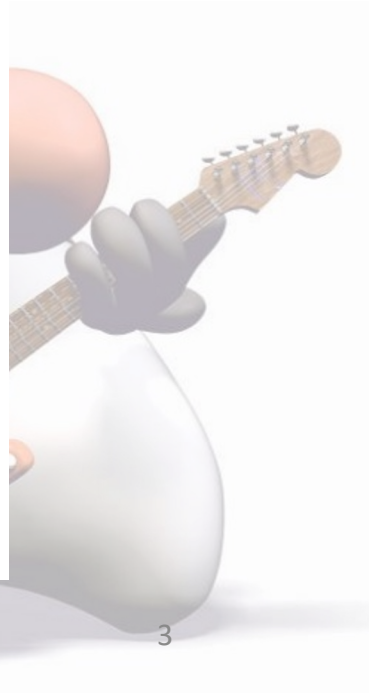

HomoErgaster
+makeTools()
+walkOn2Feet()
HomoErectus
+makeFire()
CulturalHuman
+pray()
+makeWeapons()
HomoNeanderthalensis
+surviveInIce()
+makeTools()
+dieOut()
HomoSapiens
+paintPictures()
+singASong()
+makeWeapons()
+programComputers()
+pray()

```java
public class HomoErgaster {
        public void makeTools(){
                // ...
        }

        public void walkOn2Feet(){
                //...
        }
}
```

```java
public class HomoErectus extends HomoErgaster{
        public void makeFire(){
                // ...
        }
}
```

```java
public class CulturalHuman extends HomoErectus {
        public void pray(){
                // ...
        }

        public void makeWeapons(){
                //..
        }
}
```

```java
public class HomoNeanderthalensis extends
CulturalHuman{
        @Override
        public void makeTools(){
                // ..
        }

        public void surviveInIce(){
                //..
        }

        public void dieOut(){
                // ..
        }
}
```



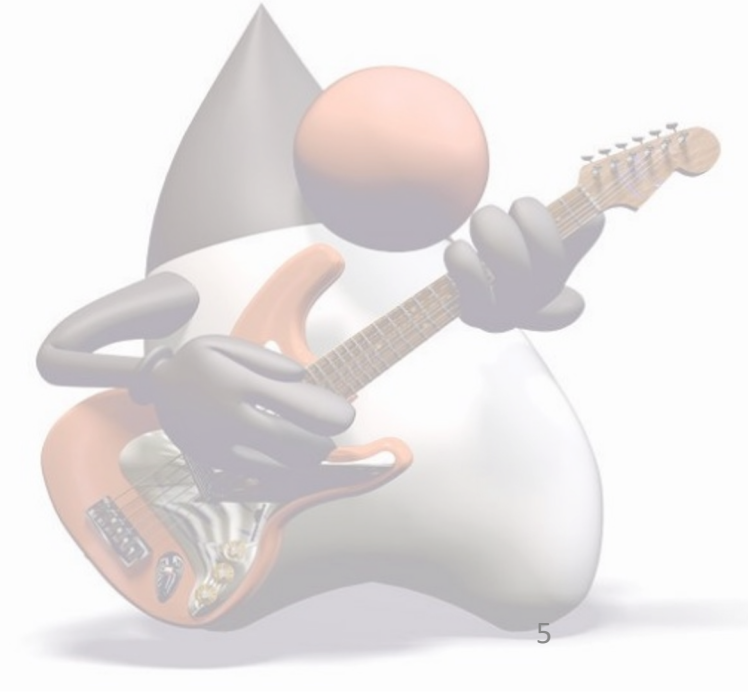
Homo Neanderthalensis kann alles, was seine Vorfahren konnten

```
public class Evolution {
	public static void main(String[] args) {
		HomoNeanderthalensis tom = new HomoNeanderthalensis();
		tom.walkOn2Feet();
		tom.makeFire();
		tom.makeTools();
		tom.makeWeapons();
		tom.pray();
		tom.surviveInIce();
	}
}
```

Wenn Klasse B von Klasse A erbt

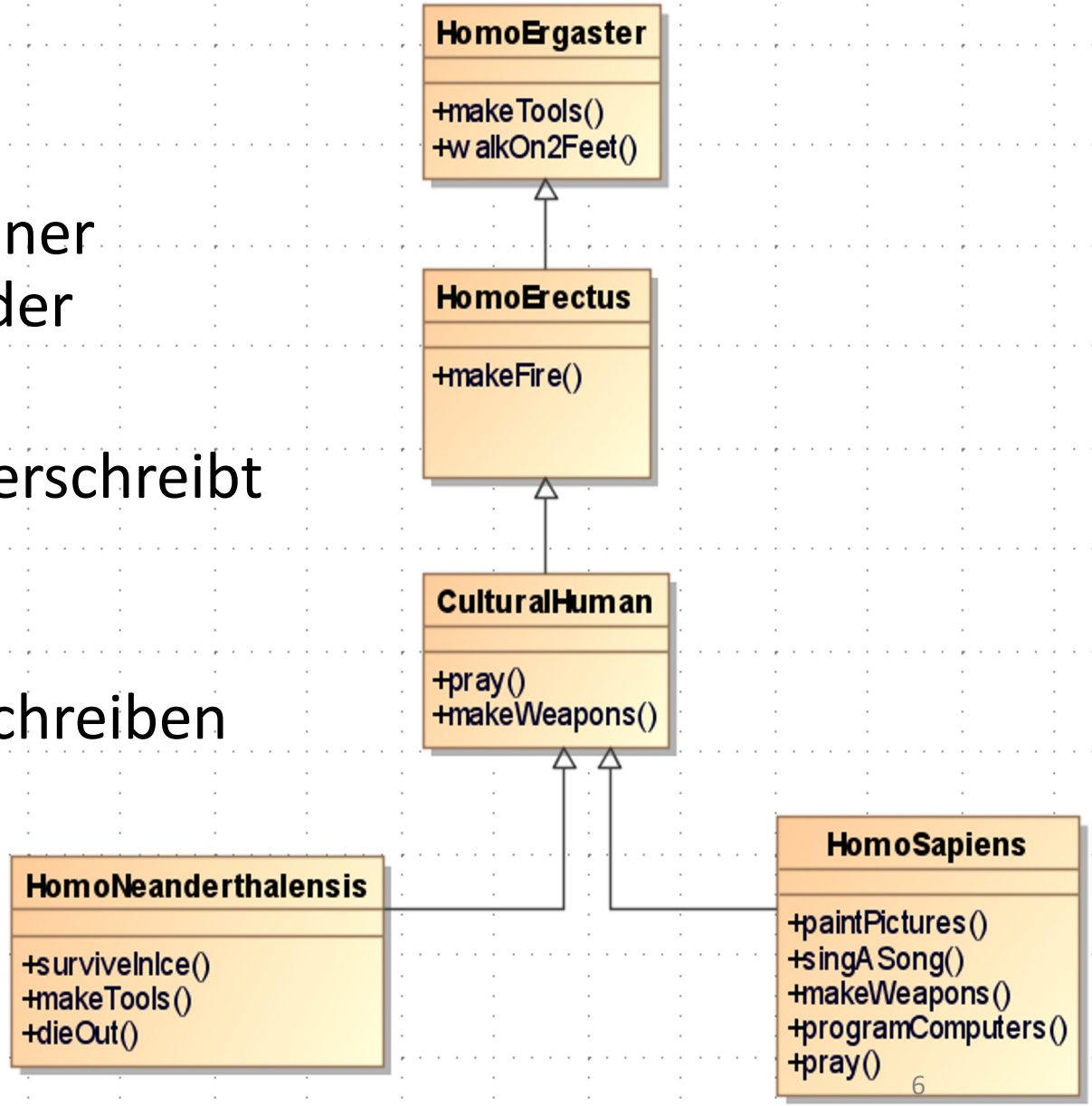
- B erbt alle Attribute von A
- B erbt alle Methoden von A
- B erbt keine Konstruktoren

- Man spricht davon das A die Superklasse von B ist
- Man spricht davon das B die Sub-Klasse von A ist
- Man kann auch sagen das B ein A ist



Überschreiben

- Die Redefinition einer Methode in einer Subklasse wird als „überschreiben“ der Methode bezeichnet.
- Beispiel: HomoNeanderthalensis überschreibt die Methode makeTools, die er von HomoErgaster geerbt hat.
- Welche anderen Beispiele für Überschreiben findest Du im rechten Beispiel?



Object ist Superklasse aller Klassen

```
public class HomoErgaster extends
Object{
        public void makeTools(){
                // ...
        }

        public void walkOn2Feet(){
                //...
        }
}
```

Ist das selbe wie…

```
public class HomoErgaster {
        public void makeTools(){
                // ...
        }

        public void walkOn2Feet(){
                //...
        }
}
```

Wichtige Object Methoden

public String toString();

public boolean equals();

Eine Object Variable kann alle Instanzen egal welchen Klassentyps halten

```
Object object = new HomoErgaster();
Object o2 = new Account();
```

Jede Klasse hat eine Implementierung von toString() – geerbt von Object

```
public void println(Object o){
        String printString = o.toString();
    // ...
}
```

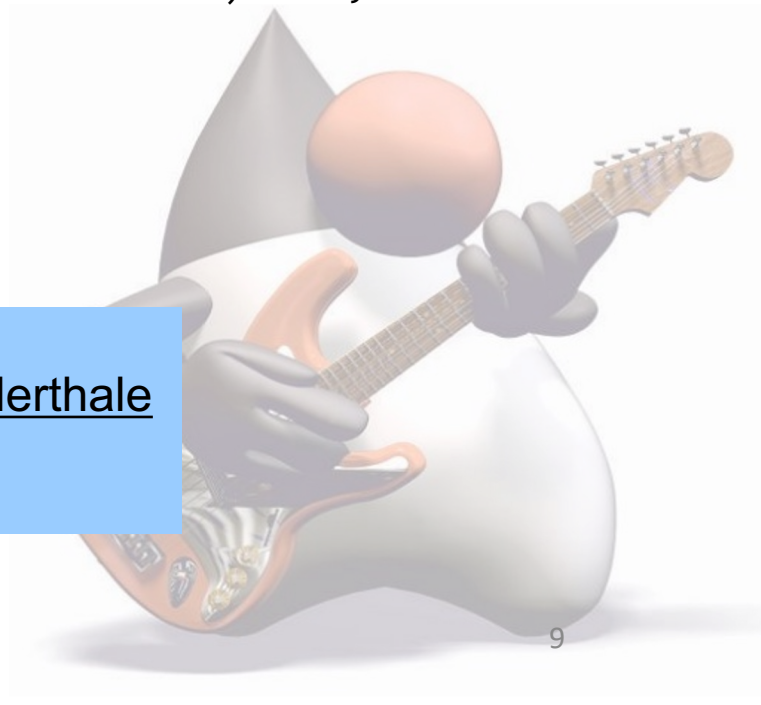
Polymorphie

Die JVM weiß immer, was der echte Typ einer Referenz ist und kann den Original Typ wiederherstellen.

```
public boolean dieOut(CulturalHuman human){
        if (human instanceof HomoNeanderthalensis){
                HomoNeanderthalensis neanderthalensis = (HomoNeanderthalensis)human;
                neanderthalensis.dieOut();
                return true;
        } else {
                return false;
        }
}
```

CulturalHuman human

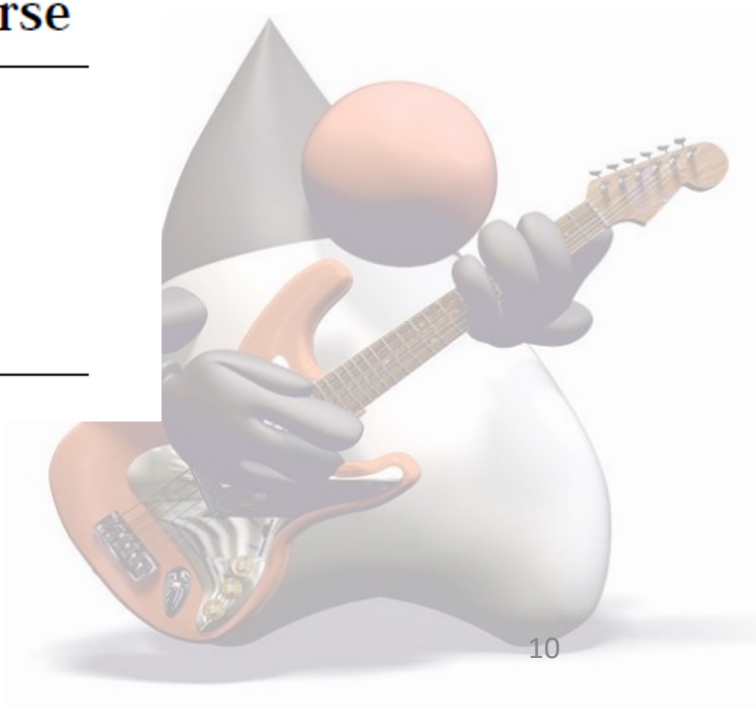
:HomoNeanderthalensis



Access Modifiers

- auf private Attribute kann eine Subklasse nicht zugreifen
- Subklassen können auf protected Variablen zugreifen

Modifier	Same Class	Same Package	Subclass	Universe
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes



Konstruktoren

```
public CulturalHuman(){
}
```

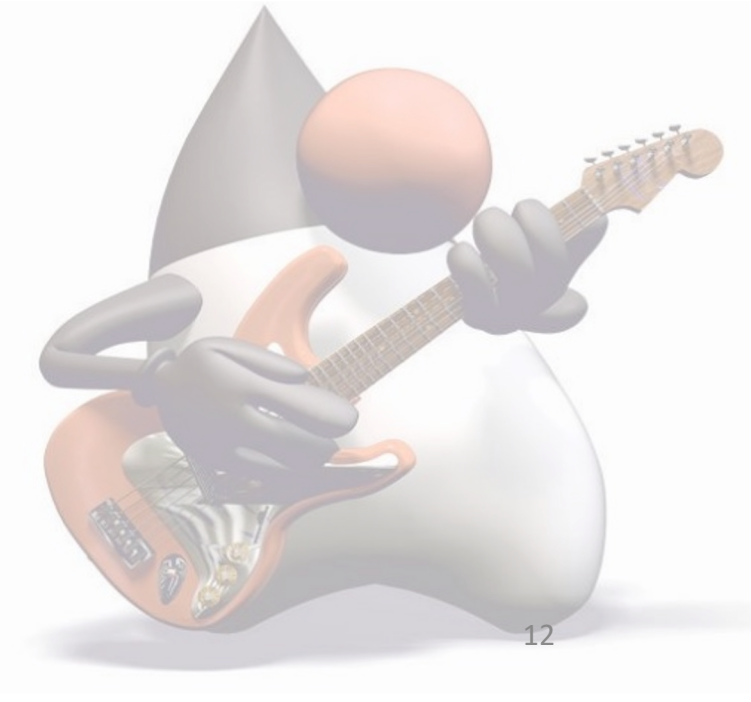
Ist eigentlich …

```
public CulturalHuman(){
        super();
}
```

- super() ruft den Konstruktor der Superklasse auf (Object)
- der super Konstruktor muss IMMER aufgerufen werden, daher muss man den Aufruf super() nicht explizit angeben
- Konstruktoren werden nicht vererbt.

Equals und HashCode

- Wann sind selbstdefinierte Objekte equals() == true ?

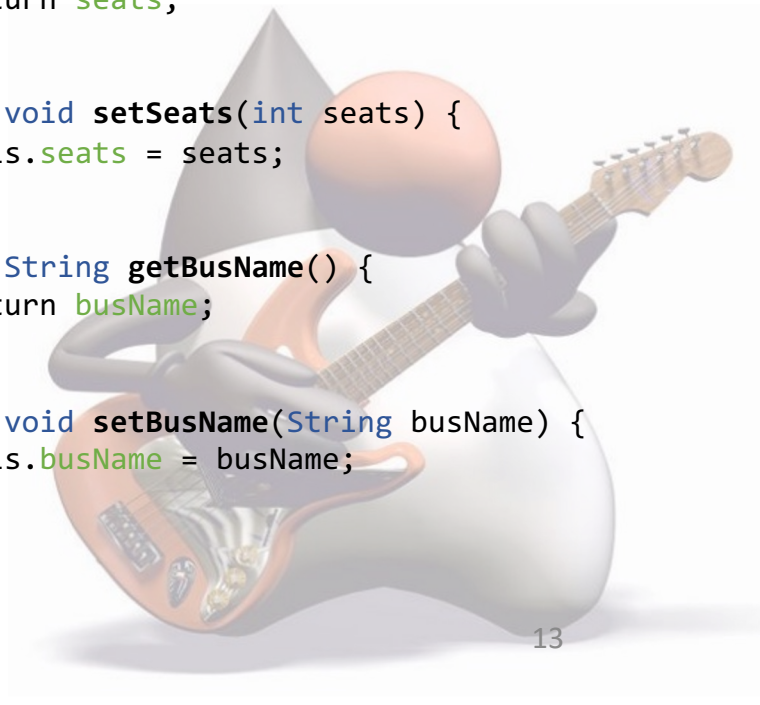


Equals und Hashcode

- Nehmen wir unsere Klasse Bus
  - Erstellen wir 2 Objekte
  - `Bus bus1 = new Bus(„MAN001“);`
  - `Bus bus2 = new Bus(„MAN001“);`
- If(bus1 == bus2) -> false; //selbe referenz?
- If(bus1.equals(bus2) -> false; //equals?

- Was müssen wir tun?

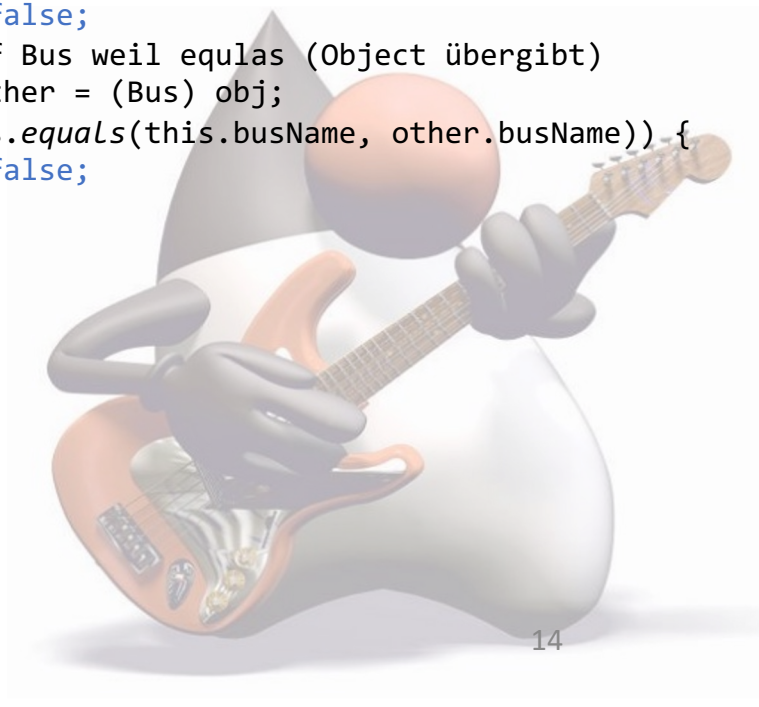
```
public class Bus extends Vehicle {

    int seats = 32;
    private String busName;

    public Bus() {
    }

    public Bus(int initialSeats) {
        super(initialSeats);
    }

    public Bus(String busName) {
        this.busName = busName;
    }

    public int getSeats() {
        return seats;
    }

    public void setSeats(int seats) {
        this.seats = seats;
    }

    public String getBusName() {
        return busName;
    }

    public void setBusName(String busName) {
        this.busName = busName;
    }
}
```



Equals und Hashcode

- Nehmen wir unsere Klasse Bus
  - Erstellen wir 2 Objekte
  - `Bus bus1 = new Bus(„MAN001“);`
  - `Bus bus2 = new Bus(„MAN001“);`
- If(bus1 == bus2) -> false; //selbe referenz?
- If(bus1.equals(bus2) -> true!; //equals?

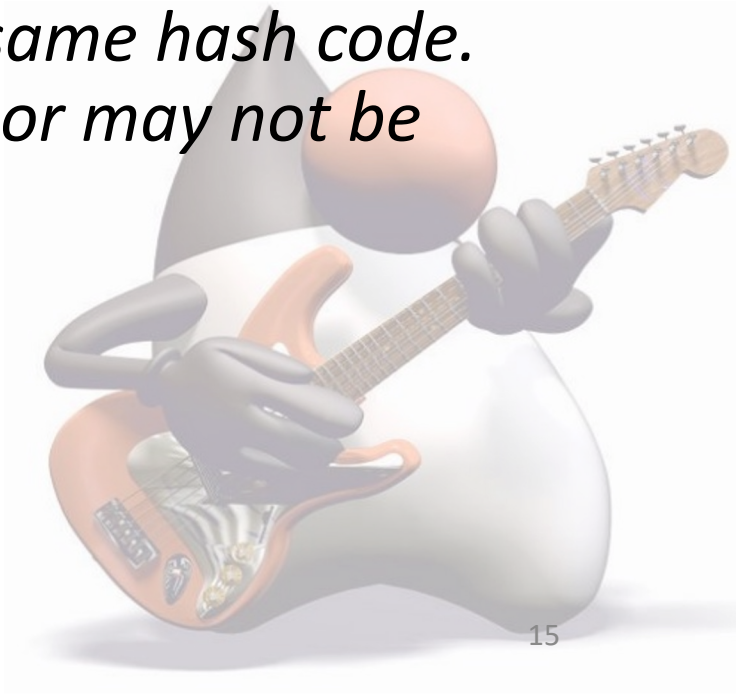
```
public class Bus extends Vehicle {

    int seats = 32;
    private String busName;
//constructors und getter / setter

 @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        } //cast auf Bus weil equlas (Object übergibt)
        final Bus other = (Bus) obj;
        if (!Objects.equals(this.busName, other.busName)) {
            return false;
        }
        return true;
    }

}
```



Wieso HashCode?

- Bestimmte Klassen aus dem Collections Framework (siehe Collections) benötige überschriebene hashCode für die korrekte und performante Funktionsweise. Stichwort: **Equals HashCode – Vertrag:**
- *The contract between equals() and hashCode() is:*
  1) If two objects are equal, then they must have the same hash code.
  2) If two objects have the same hash code, they may or may not be equal.

```
@Override
  public int hashCode() {
     int hash = 7;
     hash = 43 * hash + Objects.hashCode(this.busName);
     return hash;
  }
```



Übung 8

- Implementiere eine Klasse Sparkonto als eine Subklasse von Konto
- Sparkonto soll ein Attribut zinssatz bekommen
- Sparkonto soll die Methode abheben von Account überschreiben

- Implementieren Sie eine Klasse Gehaltskonto als eine Subklasse von Konto
- Gehaltskonto soll ein Attribut überziehungsrahmen haben.
- Überschreiben Sie die Methode abheben und erlauben Sie ein abheben bis zum Erreichen des Überziehungsrahmens