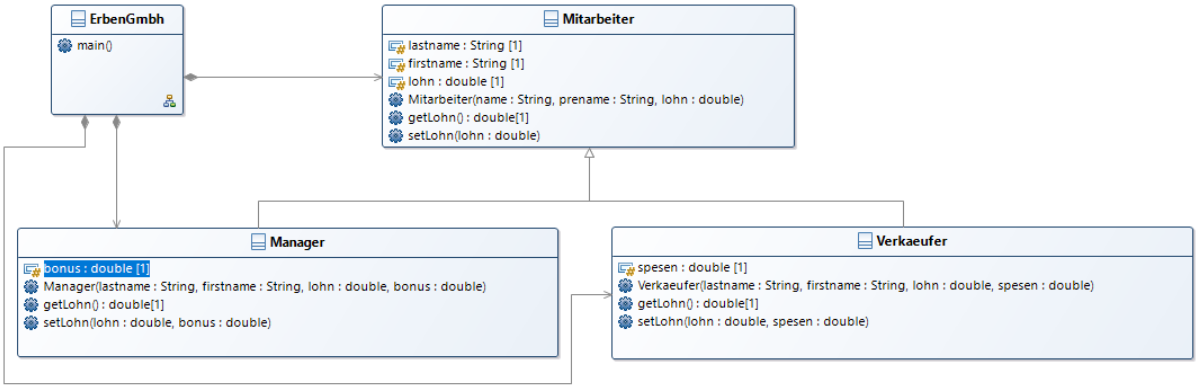
Methoden überschreiben / Override

Quelle: galileocomputing_javainsel10

Wir haben gesehen, dass eine Unterklasse durch Vererbung die sichtbaren Eigenschaften ihrer Oberklasse erbt. Die Unterklasse kann nun wiederum Methoden hinzufügen. Dabei zählen überladene Methoden – also Methoden, die den gleichen Namen wie eine andere Methode aus einer Oberklasse tragen, aber eine andere Parameteranzahl oder andere Parametertypen haben – zu ganz normalen, hinzugefügten Methoden.

Besitzt eine Unterklasse eine Methode mit dem gleichen Methodennamen und der exakten Parameterliste (also der gleichen Signatur) wie schon die Oberklasse, so überschreibt die Unterklasse die Methode der Oberklasse. Ist der Rückgabetyp void oder ein primitiver Typ, so muss er in der überschreibenden Methode der gleiche sein. Bei Referenztypen kann der Rückgabetyp etwas variieren.

Beispiel:



Von der Klasse Mitarbeiter sind die beiden Klassen Manager und Verkaeufer abgeleitet.

Der Manager hat zusätzlich das Attribut *bonus*. Sein Lohn setzt sich aus *lohn* + *bonus* zusammen.

Der Verkaeufer hat zusätzlich das Attribut *spesen*. Sein Lohn setzt sich aus *lohn* + *spesen* zusammen.

```java
package ch.bbw.pr.erbengmbh;

public class Mitarbeiter
{
   protected String lastname;
   protected String firstname;
   protected double lohn;

   public Mitarbeiter(String lastname, String firstname, double lohn)
   {
      super();
      this.lastname = lastname;
      this.firstname = firstname;
      this.lohn = lohn;
   }

   public double getLohn()
   {
      return lohn;
   }

   public void setLohn(double lohn)
   {
      this.lohn = lohn;
   }
}
```

```java
package ch.bbw.pr.erbengmbh;

public class Manager extends Mitarbeiter
{
   protected double bonus;

   public Manager(String lastname, String firstname, double lohn, double bonus)
   {
      super(lastname, firstname, lohn);
      this.bonus = bonus;
   }

   @Override
   public double getLohn()
   {
      return lohn + bonus;
   }

   //@Override -> ergibt Fehler, da es kein Override ist.
   public void setLohn(double lohn, double bonus)
   {
      super.setLohn(lohn);
      this.bonus = bonus;
   }
}
```

Zeile 9: Aufruf des Konstruktors von Mitarbeiter mittels *super*

Zeile 14: Überschreiben der Methode *getLohn*, da der Lohn eines Managers anders berechnet wird wie der Lohn eines Mitarbeiters.
Das Überschreiben wird mittels *@Override* markiert und vom Compiler überprüft.

Zeile 20: Die Methode setLohn hat neue Parameter, damit eine neue Signatur und ist damit keine überschriebene Methode.
Markiert man die Methode mit *@Override* so gibt es ein Compile-Fehler.

Zeile 22: Aufruf der Methode *setLohn* der Basisklasse mittels *super*.

Die Annotation @Override bedeutet nicht, dass diese Methode in Unterklassen überschrieben werden muss, sondern nur, dass sie selbst eine Methode überschreibt. Annotationen sind zusätzliche Modifizierer, die entweder vom Compiler überprüft werden oder von uns nachträglich abgefragt werden können. Obwohl wir die Annotation @Override nicht nutzen müssen, hat sie den Vorteil, dass der Compiler überprüft, ob wir tatsächlich eine Methode aus der Oberklasse überschreiben

Verwendung von Mitarbeiter, Manager und Verkaeufer:

```java
package ch.bbw.pr.erbengmbh;

public class ErbenGmbh
{

   public static void main(String[] args)
   {
      Mitarbeiter mitarbeiterA = new Mitarbeiter("Paul", "Muster", 6000.0);
      Manager managerB = new Manager("Rolf", "Rain", 12000.0, 2000.0);
      Verkaeufer verkaeuferC = new Verkaeufer("Volker", "Voll", 8000.0, 1500.0);

      System.out.println("Lohn von Mitarbeiter A: " + mitarbeiterA.getLohn());
      System.out.println("Lohn von Manager B: " + managerB.getLohn());
      System.out.println("Lohn von Verkaeufer C: " + verkaeuferC.getLohn());
   }
}
```

Output auf der Konsole

```
Lohn von Mitarbeiter A: 6000.0
Lohn von Manager B: 14000.0
Lohn von Verkaeufer C: 9500.0
```

Mit super an die Eltern

Wenn wir eine Methode der Basisklasse in unserer abgeleiteten Klasse überschreiben, kann es sein, dass wir die Funktionalität der Basisklasse dennoch aufrufen möchten.

Ein Beispiel ist die *toString* Methode. Wir möchten das sowohl die Angaben der Basisklasse wie auch der abgeleiteten Klasse im String enthalten sind.

Mittels des Stichwortes super kann ich explizit auf die Methode der Basisklasse zugreifen.

```
public class Garage extends Room
{
  private static final int MAX_GARAGE_SIZE = 40;

  @Override public void setSize( int size )
  {
    if ( size <= MAX_GARAGE_SIZE )
      super.setSize( size );
  }
}
```

Im Bespiel greif die überschriebene Methode setSize der Klasse Garage auf die setSize Methode der Basisklasse Room zu.

Und bei Konstruktoren?

Beispiel

```
class GameObject extends Object
{
  GameObject()
  {
    super();          // Ruft Standard-Konstruktor von Object auf
  }
}

class Player extends GameObject
{
  Player()
  {
    super();          // Ruft Standard-Konstruktor von GameObject auf
  }
}
```