

Beziehungen/Vererbung

1. „Class B extends X“. Jetzt fügen Sie eine neue Methode in X ein. Müssen Sie B anpassen?

Klasse B stehen alle nicht privaten Methoden und Felder der Klasse X zur Verfügung. In B können Sie nun darüber hinaus weitere Methoden und Felder implementieren. B ist somit Unterklasse von X und X somit Oberklasse. („B ist ein X“, wie „Audi ist ein Auto“). B erbt von X dann muss nicht angepasst werden, wenn in X schon die Methoden drin stehen die man in B braucht. Man kann aber durch Override Methoden anpassen.

```
2. Class B extends X {  
    public void newMethodinB() { .... }  
}
```

Jetzt fügen Sie eine neue public Methode in ihre abgeleitete Klasse ein. Sie möchten diese neue Methode im Code verwenden. Prüfen Sie die folgenden Codezeilen:

```
X x = new B();  
x.newMethodinB();
```

Was stellen Sie fest?

Obwohl die Methode in B ist, kann die Methode über X nicht aufgerufen werden. Die Methode aus der Subklasse kann also nicht auf das Objekt der Superklasse verwendet werden.

```
2. Class B extends X {  
    @Override public void methodinB() { .... }  
}
```

Jetzt überschreiben Sie eine Methode der Basisklasse in ihrer abgeleiteten Klasse. Sie möchten diese neue Methode im Code verwenden. Prüfen Sie die folgenden Codezeilen:

```
X x = new B();  
x.methodinB();
```

Was stellen Sie fest?

Eine geerbte Methode wird in der jeweiligen Unterklasse durch eine neue Methode überschrieben, wenn beide Methoden die gleiche Signatur besitzen. Die Signatur einer Methode besteht aus ihrem Namen und den vorgesehenen Parametertypen (in der Reihenfolge ihrer Deklaration). Die überschriebene Methode bleibt jedoch erreichbar und kann mit Hilfe des Schlüsselworts `super` auch weiterhin aufgerufen werden.

```
super.methodenname();
```

3. Versuchen Sie „Square“ von Rectangle abzuleiten (geben Sie an welche Methoden Sie in die Basisklasse tun und welche Sie in die abgeleitete Klasse tun)

Class Square extends Rectangle

Basisklasse: Rectangle, Unterklasse: Square

```
public class Rectangle {  
    private int width;  
    private int height;
```

```

        public int getHeight(){
            return height;
        }
        public int getWidth() {
            return width;
        }
        public void setHeight(int p) {
            height=p;
        }
        public void setWidth(int p) {
            width=p;
        }
    }

    public class Square extends Rectangle {
        public void setHeight(int p) {
            super.setHeight(p);
            super.setWidth(p);
        }
        public void setWidth(int p){
            this.setHeight(p);
        }
    }
}

```

4. Jetzt machen Sie das Gleiche umgekehrt: Rectangle von Square ableiten und die Methoden verteilen.

```

public class Square {
    private int width;
    private int height;

    public int getHeight(){
        return height;
    }
    public int getWidth() {
        return width;
    }
    public void setHeight(int a) {
        height=a;
    }
    public void setWidth(int b) {
        width=b;
    }
}

```

5. Nehmen Sie an, „String“ wäre in Java nicht final. Die Klasse Filename „extends“ die Klasse String. Ist das korrekt? Wie heißt das Prinzip dahinter?

Das Prinzip heißt: Liskovsches Substitutionsprinzip

Thread-Sicherheit: Immutables sind immer thread-sicher, weil ein Thread sie erst komplett bauen muss, bevor sie an einen anderen übergeben werden können - und nach dem Bauen können sie nicht mehr verändert werden.

Außerdem wollten die Erfinder der Java-Laufzeitumgebung immer einen Fehler auf der Seite der Sicherheit machen. Die Möglichkeit, String zu erweitern, kann ein ganzes Wespennest öffnen, wenn man nicht weiß, was man tut.