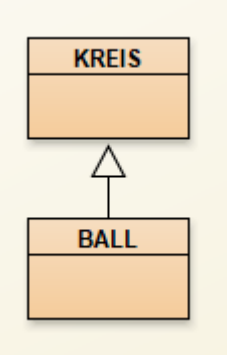
Vererbung

Durch Vererbung kann schon bestehender Code (für z. B. Grafik usw.) genutzt und für die eigenen Zwecke erweitert werden.

```
public class BALL extends KREIS{
    public BALL(){
        super();
    }
}
```



- Mittels des Schlüsselworts **extends** kann von einer Klasse geerbt werden.

  KREIS ist hier jetzt die Superklasse und BALL die Subklasse. Durch das Erben hat der BALL **alle** Eigenschaften und Fähigkeiten (Attribute und Methoden) der Superklasse KREIS **erhalten**. Der BALL **ist** somit ein KREIS.

- Der erste Befehl im Konstruktor in einer Subklasse muss der Aufruf eines Konstruktors der Superklasse sein. Gegebenenfalls müssen dem Superkonstruktor Werte für vorhandene Parameter mitgegeben werden.

  Mit dem Schlüsselwort **this** kann auf die Attribute und Methoden dieser Klasse (hier: BALL) zugegriffen werden.
  Mit dem Schlüsselwort **super** kann auf die Attribute und Methoden der Superklasse (hier: KREIS) zugegriffen werden.

Überschreiben von Methoden

Durch das Überschreiben von Methoden aus der Superklasse können die geerbten Methoden für die eigene Klasse angepasst werden.

Die Signatur der Methode aus der Superklasse wird dabei exakt übernommen und eine Zeile darüber wird ein **@Override** geschrieben. Beispiele:

```
@Override
public void tasteReagieren(int taste){
    ...
}
@Override
public void tick(){
    ...
}
```

Im Methodenrumpf (zwischen den geschweiften Klammern) kann die Methode jetzt für die eigenen Zwecke angepasst selbst geschrieben werden.

Bedingte Anweisung und Verzweigung

Manchmal möchtest du einen Befehl nur ausführen, wenn eine bestimmte Bedingung erfüllt ist. So eine Bedingung ist meist ein Vergleich. Für Vergleiche stehen dir folgende Operatoren zur Verfügung:

• Gleichheit **==**

alter == 20 name == “Lisa“ volljaehrig == true figur.beruehrt() == true/false

• Ungleichheit **!=** **<** **>** **<=** **>=**

alter < 18 alter >= 18 name != “Sepp“ !figur.beruehrt()

Ein Vergleich liefert immer eine Antwort vom Datentyp boolean. (true / false)

Bedingte Anweisung: WENN . . . DANN

Hin und wieder soll ein Befehl nur unter einer bestimmten Bedingung ausgeführt werden:

```
WENN    if ( this.besitzer == “Sepp“ ) {
DANN    auto.fahren ();
        }
```

Fallunterscheidung: WENN . . . DANN . . . SONST

Das letzte Beispiel kann man auch um eine alternative Anweisung erweitern.

```
WENN    if ( !karol.istWand() ) {
DANN    karol.schritt();
        } else {
SONST   karol.linksDrehen();
        karol.schritt();
        }
```

Mehrfache Fallunterscheidung: WENN . . . DANN . . . SONST WENN . . .

Man kann das Beispiel auch noch weiter ausdehnen:

```
WENN          if ( taste == TASTE.RAUF ) {          // Pfeil rauf
DANN          spielfigur.oben();
SONST WENN    } else if ( taste == TASTE.RECHTS ) {  // Pfeil rechts
DANN          spielfigur.rechts();
SONST WENN    } else if ( taste == TASTE.RUNTER ) {  // Pfeil runter
DANN          spielfigur.unten();
SONST WENN    } else if ( taste == TASTE.LINKS ) {   // Pfeil links
DANN          spielfigur.links();
              }
```