1

Merkblatt

Klassen und Objekte

Klassen sind Vorlagen für Objekte.

1 Aus einer Klasse kann man beliebig viele Objekte herstellen.

Objekte haben

- **Attribute**
- **Konstruktoren**
- **Methoden**

2 Attribute bestimmen die **Eigenschaften** eines Objektes

3 Konstruktoren geben an, was beim **Erzeugen** eines neuen Objektes passieren soll.

4 Methoden bestimmen das **Verhalten** eines Objektes.

1
```
public class Gegenstand
{
   String name;
2  int angriffswert, goldwert, zustand;

   public Gegenstand(String n, int a, int g)
3  {
      name = n;
      angriffswert = a; goldwert = g;
      zustand = 100;
   }

   public int gibAngriffswert()
4  {
      return angriffswert;
   }

   public void beschaedige(int schaden)
4  {
      zustand = zustand - schaden;
      if (zustand < 0)
        zustand = 0;
   }
}
```

Merkblatt

Methoden

Man unterscheidet:

- **sondierende Methoden** 1
- **manipulierende Methoden** 2

Unabhängig davon gibt es:

- **private Methoden** 3
- **öffentliche Methoden** 4

Außerdem gibt es

- **Methoden ohne Parameter** 5
- **Methoden mit Parametern** 6

```
public class Gegenstand
{
   String name;
   double angriffswert, goldwert, zustand;

   public Gegenstand(String n, int a, int g)
   {
      name = n;
      angriffswert = a; goldwert = g;
      zustand = 100.0;
   }

   public int gibAngriffswert()
   {
      return angriffswert;
   }

   private void werteBerechnen(double faktor)
    {
        angriffswert *= faktor;
        goldwert *= faktor;
    }

    public void beschaedigen(int prozent)
    {
        double faktor = (100.0-prozent)/100.0;
        werteNeuBerechnen(faktor);
    }
}
```

2

1 4 5

2 3 6

2 4 6

Merkblatt

Sondierende Methoden

Sondierende Methoden verändern keine Attributwerte

- **Get-Methoden** liefern jeweils einen Attributwert zurück (1)
- Andere sondierende Methoden berechnen Ergebnisse aus den Attributwerten und liefern die Ergebnisse zurück. (2)
- Der **return-Befehl** beendet die Methode und liefert dabei den hinter ihm stehenden Wert an die aufrufende Methode zurück. (3)

(4)

```
public class Gegenstand
{
   String name;
   double angriffswert, goldwert, zustand;

   public Gegenstand(String n, int a, int g)
   {
      name = n;
      angriffswert = a; goldwert = g;
      zustand = 100.0;
   }

1  public int gibAngriffswert()
   {
      return angriffswert;
   }

2  private double gibKampfkraft()
   {
      double k = gibAngriffswert() * zustand;
3     return k / 10;
      System.out.println("Wird nie gezeigt!");
   }
}
```

3

4

Dieser Befehl kommt nach dem return-Befehl, daher kommt er nie zur Ausführung.

gibKampfkraft() ist die aufrufende Methode für gibAngriffswert().

Der Rückgabewert einer sondierenden Methode kann wie eine Variable eingesetzt werden!

4

Merkblatt

Manipulierende Methoden

Manipulierende Methoden

verändern bestimmte Attributwerte

- **Set-Methoden** setzen bestimmte Attributwerte zurück. (1)
- Andere manipulierende Methoden verändern mehrere Attributwerte. (2)

Attributwerte sollten nur mit Hilfe von manipulierenden Methoden verändert werden!

```
public class Gegenstand
{
   String name;
   double angriffswert, goldwert, zustand;

   public Gegenstand(String n, int a, int g)
   {
      name = n;
      angriffswert = a; goldwert = g;
      zustand = 100.0;
   }

   public int setzeAngriffswert(double neu)
   {
      angriffswert = neu;
   }

   private void werteBerechnen(double faktor)
   {
        angriffswert *= faktor;
        goldwert *= faktor;
   }

   public void ausgeben()
   {
      System.out.println
      (name + ":" + angriffswert);
   }
}
```

(1)

(2)

Merkblatt

Parameter

Parameter sind Mittel, mit denen Methoden Informationen von außen bekommen.

Eine Methode kann mehrere **Parameter** haben.

Für jeden Parameter muss der **Datentyp** angegeben werden. Die Parameter werden durch **Kommata** getrennt.

Es gibt auch Methoden ohne Parameter. Dann ist die **Parameterliste** leer ( ).

Parameter sind nur innerhalb der jeweiligen Methode definiert. Außerhalb der Methode existieren sie nicht.

```
public class Gegenstand
{
   String name;
   double angriffswert, goldwert, zustand;

   public Gegenstand(String n, int a, int g)
   {
      name = n;
      angriffswert = a; goldwert = g;
      zustand = 100.0;
   }

   public int setzeAngriffswert(double neu)
   {
      angriffswert = neu;
   }

   private void werteBerechnen(double faktor)
   {
        angriffswert *= faktor;
        goldwert *= faktor;
   }

   public void ausgeben()
   {
      System.out.println(faktor);
   }
}
```

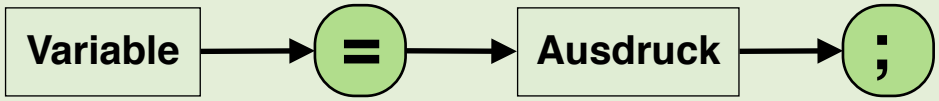
Hier gültig!

Hier nicht gültig: Fehlermeldung!

Merkblatt

Zuweisungen

Zuweisungen sind Anweisungen, bei der einer Variablen ein Wert zugewiesen wird.



Variable kann ein Attribut oder eine lokale Variable sein.

Links darf immer nur <u>eine</u> Variable stehen!

Ausdruck kann eine Zahl, eine Berechnung oder der Rückgabewert einer sondierenden Methode sein.

```
public class Held
{
   String name;
   double kraft, leben;

   public Held(String n, int k)
   {
      name = n;
      kraft = k;
      leben = 100.0;
   }

   public double gibKraft()
   {
      return kraft;
   }

   public void berechn ()
   {
       double angriff;

      angriff = gibKraft() * leben / 100;
   }
```

Variable ist ein Attribut.

Variable ist eine lokale Variable.

Ausdruck ist hier eine Berechnung.

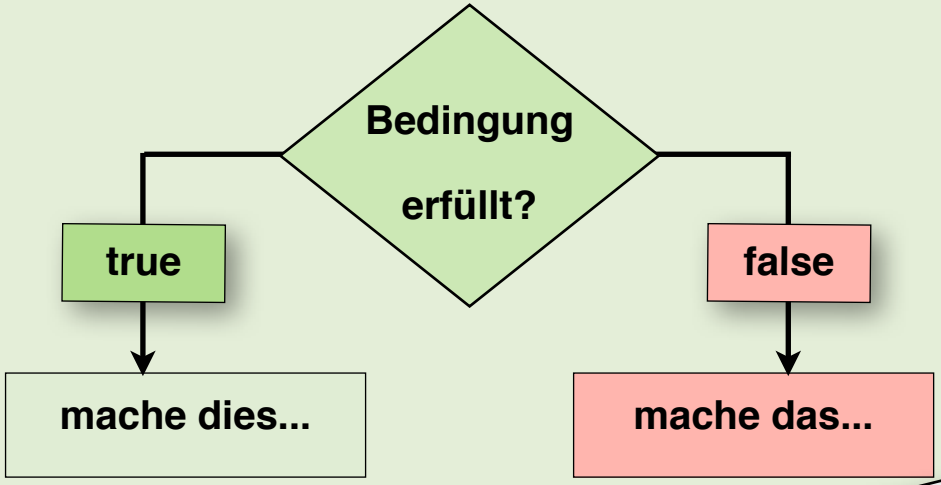
Der Rückgabewert einer sondierenden Methode ist Teil der Berechnung.

Merkblatt

Bedingungen

if-else-Bedingungen dienen zur

Implementation der zweiseitigen Auswahl!




```
public class Held
{
   String name;
   double kraft, leben;

   public Held(String n, int k)
   {
      name = n;
      kraft = k;
      leben = 100.0;
   }

   public double gibKraft()
   {
      if (leben > 0)
         return kraft;
      else
      {
         System.out.println("Der Held ist tot!");
         return 0;
      }
   }
}
```

Bedingung erfüllt! **true**

Bedingung nicht erfüllt! **false**

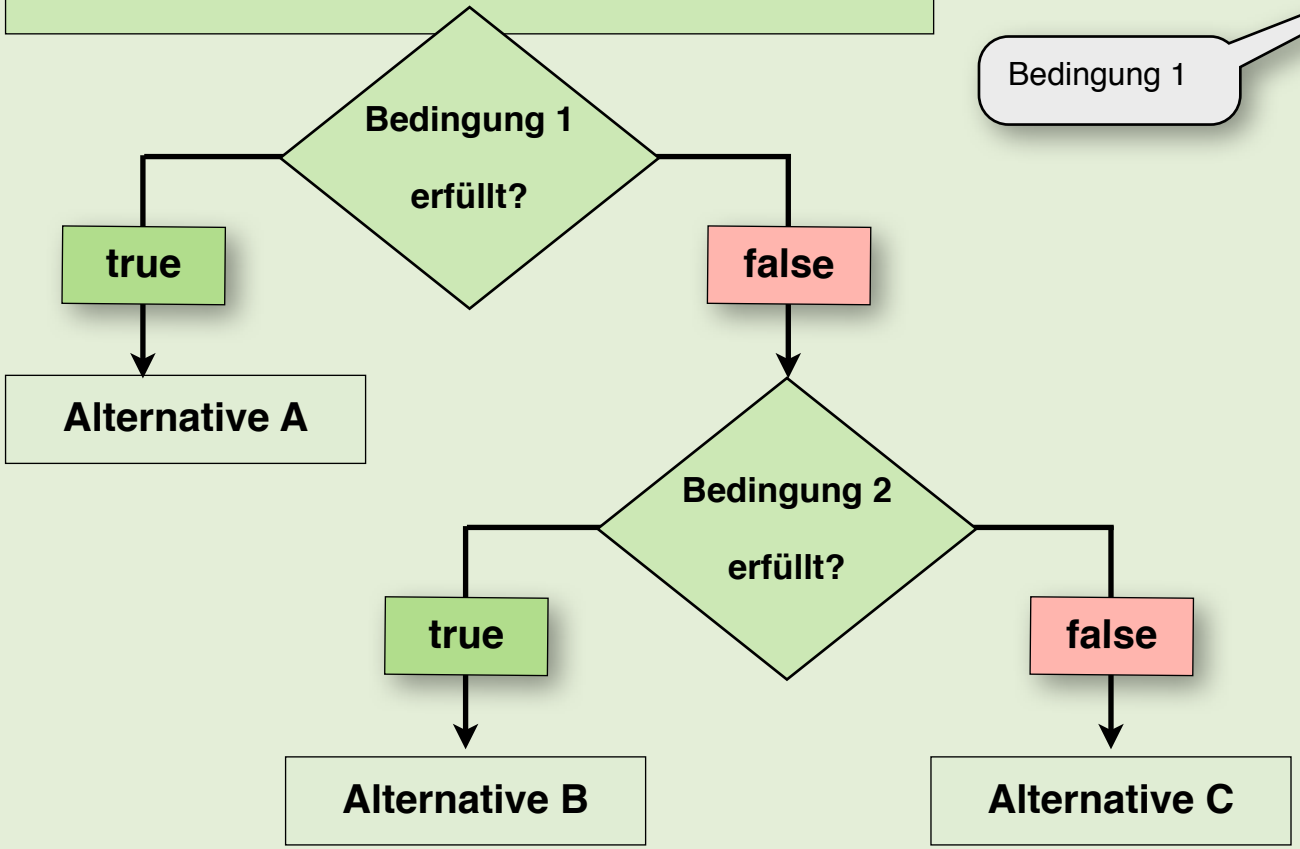
einfache Anweisung

mehrere Anweisungen müssen durch { } zusammengefasst werden.

Merkblatt

komplexere Bedingungen

geschachtelte if-else-Bedingungen dienen zur Implementation der mehrseitigen Auswahl!




```
public int steuersatz(int einkommen)
{
   if (einkommen <= 1000)
      return 0;
   else
      if (einkommen <= 10000)
         return 12;
      else
         return 15;
}
```

Bedingung 1

Bedingung 2

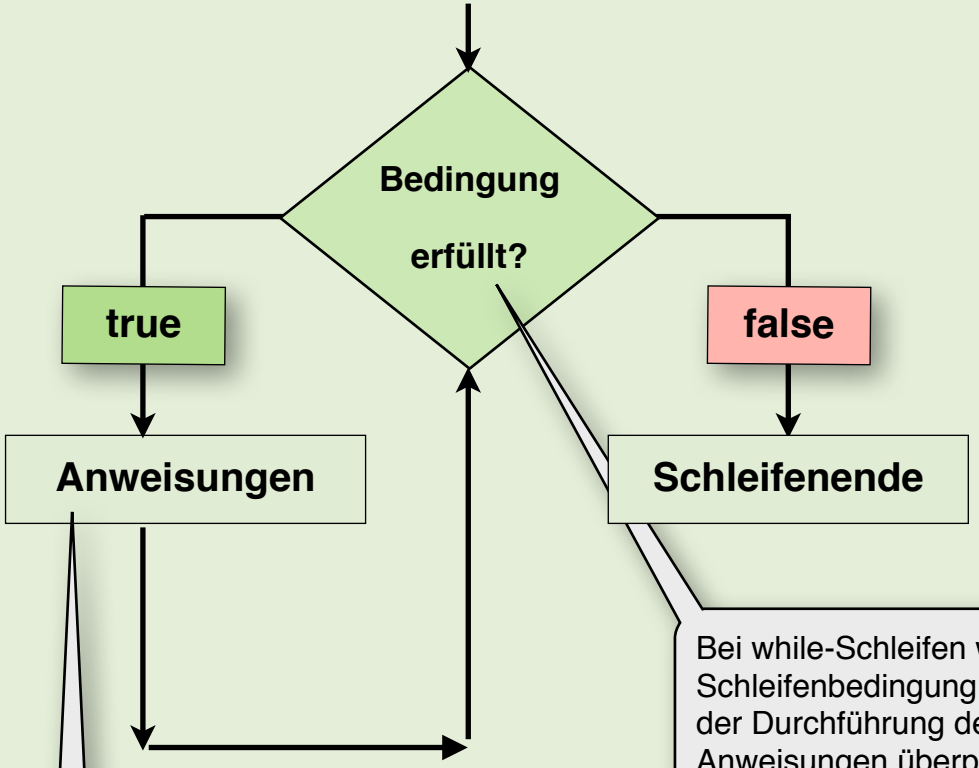
```
public int steuersatz(int einkommen)
{
   if (einkommen <= 1000)
      return 0;
   else if (einkommen <= 10000)
       return 12;
   else
       return 15;
}
```

Etwas übersichtlichere Schreibweise

Merkblatt

while-Schleifen

while-Schleifen sind vorprüfende Schleifen!




```
public void fahren(int strecke)
{
   int geschafft = 0;

   while (geschafft < strecke)
   {
      geschafft++;
      System.out.println(geschafft + " km geschafft");
   }
   System.out.println ("Ziel erreicht!");
}
```

Der Wert wächst bei jedem Schleifendurchgang, damit irgendwann die Schleifen-Bedingung nicht mehr gilt und die Schleife terminiert.

Bedingung

Anweisungen

Schleifenende

wird nach dem Schleifenende ausgeführt.

Bei while-Schleifen wird die Schleifenbedingung VOR der Durchführung der Anweisungen überprüft.

Es kann passieren, dass die Schleife NIE durchlaufen wird!

Die Anweisungen müssen dafür sorgen, dass die Bedingung irgendwann nicht mehr erfüllt ist. Sonst: **Endlosschleife**!

Merkblatt

for-Schleifen

for-Schleifen sind Zähl-Schleifen: Die Zahl der Schleifendurchgänge wird zu Beginn der Schleife festgelegt. Man muss also wissen, wie viele Durchgänge erforderlich sind!

Eine for-Schleife besteht aus

- einer **Laufvariablen** mit
- einem **Startwert**
- einer **Schleifenbedingung**
- einer **Inkrementation** oder
- einer **Dekrementation** der Laufvariable.

```
public void fahren(int strecke)
{
   for (int i=1; i<= strecke; i++)
   {
      geschafft++;
      System.out.println(geschafft + " km geschafft");
   }
   System.out.println ("Ziel erreicht!");
}

public void rueckwaerts(int bisWohin)
{
   int sum = 0;
   for (int k=100; k > -50; k = k-3)
      sum = sum + k;
}
```

Die **Laufvariable** muss nicht unbedingt i heißen.

Dekrementation: mit einer for-Schleife kann man auch rückwärts zählen!

man kann auch andere Werte als 1 verwenden.