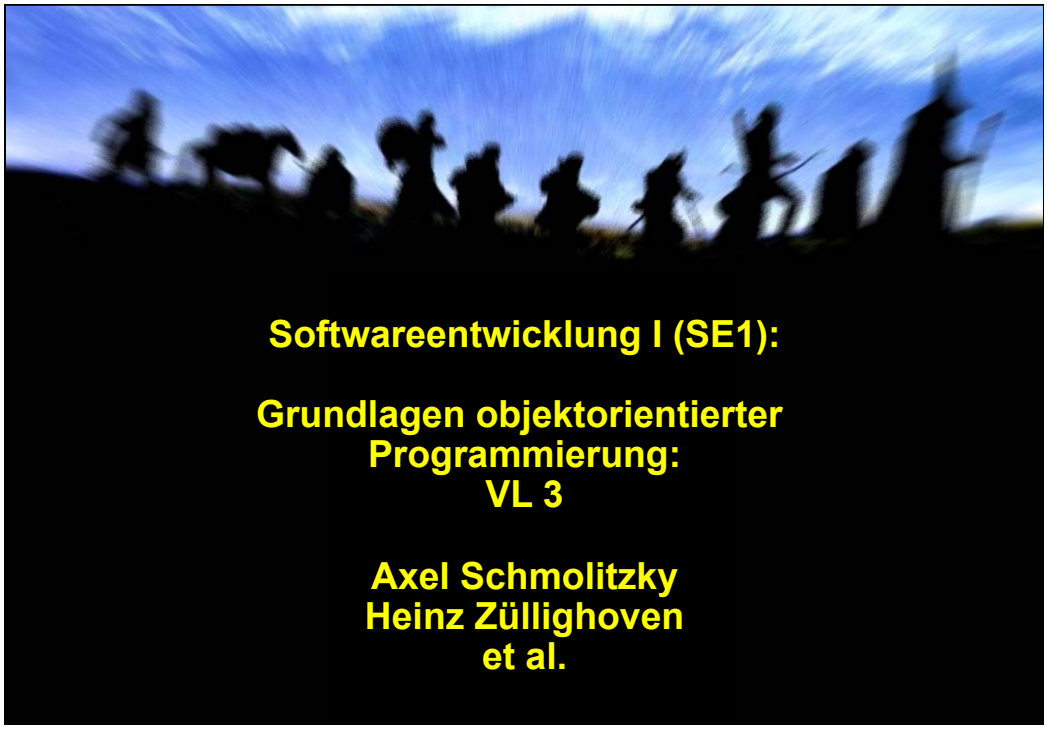
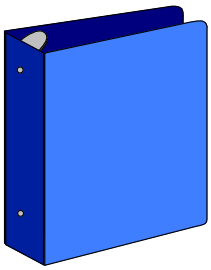



## Modellierung von Gegenständen des Anwendungsbereichs



**Ordner**
- Dokument entnehmen
- Dokument einfügen
- mit Text beschriften
- Dokument auswählen
- das Register aufschlagen
- voll?

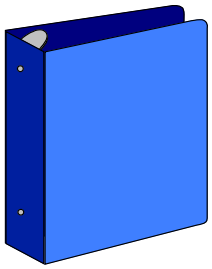
...

**Modellierung des Gegenstands**
- Begriff als Substantiv
- Umgangsform als Verb

**Das Konzept der fachlichen Modellierung von Gegenständen:**
- Wähle anwendungsfachliche Begriffe.
- Beschreibe Umgangsformen und die dabei benötigten weiteren Gegenstände



## Unsere Grundidee



Ein Gegenstand...

**Mein Kundenordner**

einfügen
beschriften
suchen
entnehmen

... wird in seinem Verhalten fachlich modelliert ...

**Klasse Ordner**

einfügen
beschriften
suchen
entnehmen

```
einfügen (d : Dokument) is
  do
     pos := current_pos
     docs.insert (d,pos)
...
```

Liste der Dokumente
Zeichenkette
Integer

... und softwaretechnisch als Objekt in einer Klasse beschrieben.



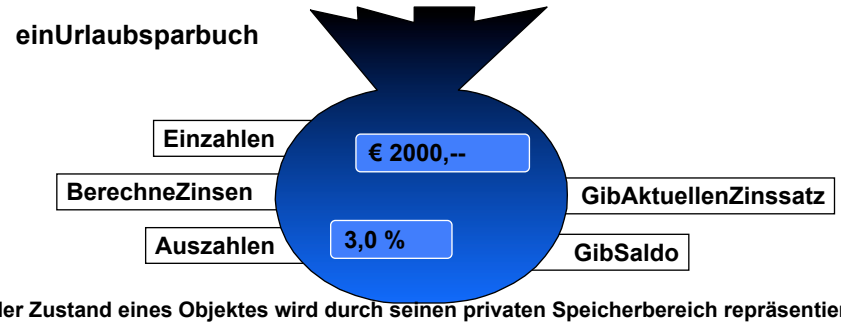
 

## Alltägliche Umgangsformen mit Gegenständen

- Die Art und Weise, wie mit Gegenständen im Rahmen der verschiedenen Aufgaben gearbeitet wird.
  Wir untersuchen:
  - Welche **Informationen** werden an den Gegenständen "abgelesen"?
  - Welche **Veränderungen** werden an den Gegenständen vorgenommen und welche **Aktionen** werden ausgelöst, ohne dass sie zerstört oder in andersartige Gegenstände transformiert werden?



## Beispiel: Vom Gegenstand zum Objekt




- **der Zustand eines Objektes wird durch seinen privaten Speicherbereich repräsentiert**
- **der private Speicherbereich eines Objektes ist von außen weder direkt zugreifbar noch direkt veränderbar**
- **dem Objekt sind vielmehr Operationen zugeordnet, mit denen sein Zustand sondiert und manipuliert werden kann**
- **das Resultat einer Operation hängt von ihren aktuellen Parametern sowie dem Zustand des Objektes ab**



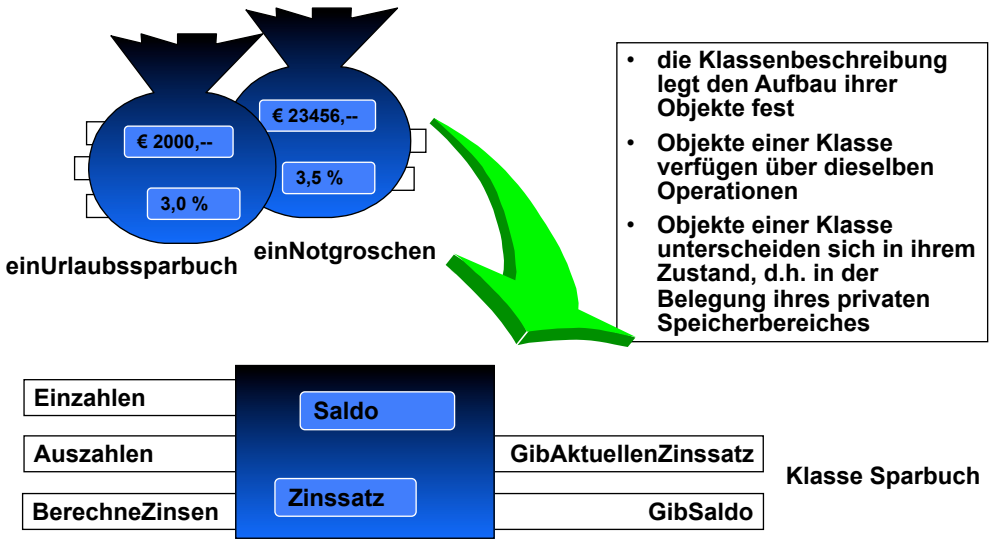
 

## Von den Gegenständen zu Objekten

- Objekt:
  - **Fachlich**
    - Objekte entsprechen den für die *Anwendung relevanten Gegenständen*. Die Gegenstände sind charakterisiert durch die Art und Weise, wie mit ihnen gearbeitet wird. Leitfragen:
      - **Welche *Informationen* wird an ihnen gesehen?**
      - **Welche *Veränderungen* können an ihnen vorgenommen werden, ohne daß sie zerstört oder in andersartige Gegenstände transformiert werden?**
      - **Welche *Aktionen* können an ihnen ausgelöst werden?**
  - **Technisch**
    - Objekte sind die *Komponenten des Systems*. Ein Objekt ist systemweit eindeutig *identifizierbar* und hat einen *Zustand*, der in einem privaten Speicherbereich des Objekts repräsentiert ist. Dem Objekt sind *Operationen* zugeordnet, die Informationen über das Objekt liefern und den Zustand des Objektes verändern können. Der Zustand eines Objektes kann nur mit Hilfe dieser Operationen gelesen oder verändert werden.



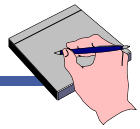
## Beispiel: Von den Objekten zu Klassen





## Unsere erste selbst geschriebene Klassendefinition



```
class Girokonto
{
  private int _saldo;

  public void einzahlen( int betrag )
  {
    _saldo = _saldo + betrag;
  }
}
```

- Ein Java-Programm besteht aus Textdateien.
- In jeder Textdatei ist eine Klasse beschrieben.
- Die textuelle Beschreibung einer Klasse nennen wir **Klassendefinition**.
- Wir bearbeiten Klassendefinitionen mit einem **Editor**.



## Von den Begriffen zu Klassen

- Klasse:
  - **Fachlich**
    - *Gegenstände*, die wir als *gleichartig* ansehen, bringen wir "auf den Begriff". In *Begriffsbeschreibungen* wird unser Verständnis von Gegenständen des Anwendungsfeldes wiedergegeben. Fachliche Begriffe sind die *Grundlage* der Bildung von *Klassen*. Das Verständnis von Klassen ist wie die Begriffsbildung *personenabhängig*. So wie sich unser Verständnis von einem Anwendungsbereich verändert, ändern sich auch die Begriffe und damit die Klassen.
  - **Technisch**
    - Klassen sind Texte, die Objekte beschreiben. Diese Beschreibungen dienen als "Erzeugungsmuster" für die Objekte der Klasse, d.h. sie definieren die Eigenschaften von Objekten. Dies umfaßt die den Objekten zugeordneten *Operationen* und deren interne Realisierung durch Algorithmen und Datenstrukturen.



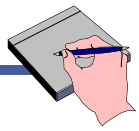
 

## Klassen als Basis

- Klassen sind die Basis objektorientierter Programmierung.
  - Eine Klasse ist eine **statische Beschreibung** eines fachlichen oder technischen Konzepts.
  - Eine Klasse beschreibt die **Eigenschaften ihrer Exemplare**.
  - Eine Klasse ist eine Art Schablone oder „Fabrik“, mit deren Hilfe **Exemplare** der Klasse **erzeugt** werden.
  - Die Klassendefinition - die statische textuelle Beschreibung einer Klasse - **bleibt bestehen**, während die Exemplare einer Klasse nur für die Dauer der Ausführung eines Programms existieren.



## Klassen als Schablonen/Erzeugungsmuster für Exemplare



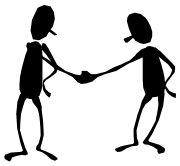
| Girokonto |
| --- |
| _dispo : Betrag<br>_saldo : Betrag |
| istAuszahlenMöglich(b:Betrag) : Boolean<br>auszahlen(b:Betrag) |

- Als **Exemplar** bezeichnet man das aus einer **Klasse** erzeugte **Objekt**.
- Eine Klasse definiert somit das **prinzipielle Verhalten** aller ihrer Exemplare.
- Von einer Klasse können **beliebig viele** Exemplare erzeugt werden.
- Aber: Jedes Exemplar hat einen **eigenen Zustand**, der verändert werden kann, und kann deshalb anders auf **dieselbe Anfrage** reagieren.



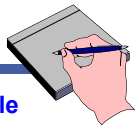
## Zentral bei jeder Software: Dienstleister und Klienten

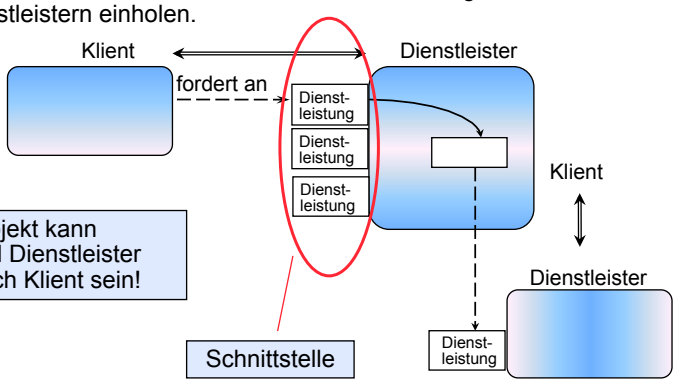
- Das Objekt, das bei einer bestimmten (Teil-)Aufgabe einen Dienst leistet, ist der **Dienstleister**.
- Das Objekt, das eine konkrete Dienstleistung eines anderen Objektes in Anspruch nimmt, wird als **Klient** bezeichnet.



## Dienstleistungen an der Schnittstelle



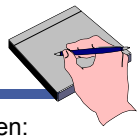
- Objekte bieten **Dienstleistungen** als **Methoden** an ihrer **Schnittstelle** an.
- Diese Dienstleistungen werden von anderen Objekten, den Klienten, benutzt. Dazu fordert der Klient eine Dienstleistung des Anbieters an.
- Der Anbieter kann selbst wieder Teile seiner Dienstleistung von anderen Dienstleistern einholen.





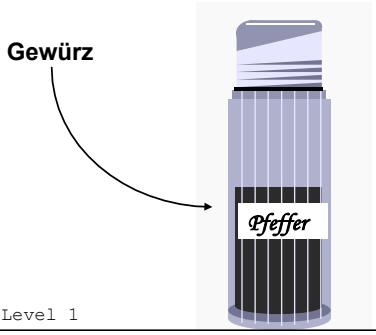

 

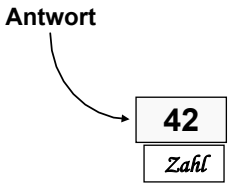
## Imperative Variablen



Der Begriff **Variable** ist grundlegend für das Verständnis imperativer Sprachen:

- Eine Variable ist eine **Abstraktion eines physischen Speicherplatzes**.
- Sie hat einen **Namen** (häufig auch: **Bezeichner**), über den sie angesprochen werden kann.
- Eine **Variable** hat den Charakter eines Behälters:
  - Sie hat eine **Belegung** (ihren aktuellen Inhalt), die sich **ändern** kann;
  - und einen **Typ**, der Wertemenge sowie zulässige Operationen und weitere Eigenschaften festlegt.

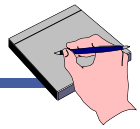



Die Typen sind hier *Pfeffer* und *Zahl*.



## Deklaration und Initialisierung



- **Vor der Verwendung** einer Variablen in imperativen Programmiersprachen muss sie bekanntgemacht, d.h. **deklariert** werden.
- Vereinfacht geschieht dies durch:
  - Angabe des **Typs**,
  - Vergabe eines **Namens** über einen **Bezeichner** (engl.: identifier).
- Durch die **reine Deklaration** von Variablen ist deren **Belegung** zunächst meist **undefiniert**.
- Erst bei der **Initialisierung** wird eine Variable erstmalig mit einem gültigen Wert befüllt.

*Deklaration*

```
int i;
boolean b;
```

*Deklaration und Initialisierung*

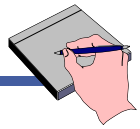
```
int i = 42;
boolean b;

b = true;
```



## Merkmale unserer ersten Klasse



```
class Girokonto
{
  private int _saldo;

  public void einzahlen( int betrag )
  {
    _saldo = _saldo + betrag;
  }
}
```

- Java-Programme bestehen aus **Klassen** (hier: `Girokonto`).
- Die Klasse definiert eine **Methode** (hier: `einzahlen`).
- Die Methode erhält einen Parameter (hier: `betrag` vom Typ `int`) und hat keinen Rückgabewert (hier: Schlüsselwort `void`).
- Im Rumpf der Methode wird ein Wert einem **Zustandsfeld** zugewiesen (hier: `_saldo`).
- Das Feld muss **deklariert** sein (hier vom Typ `int`).
- Alternativ nennen wir die Felder in einer Klassendefinition auch **Exemplarvariablen**.



## Abgleich mit den Prinzipien der Objektorientierung

```
class Girokonto
{
  private int _saldo;

  public void einzahlen( in
  {
    _saldo = _saldo + betra
  }
}
```

- Das Verhalten eines Objekts ist durch seine angebotenen Dienstleistungen (Methoden) bestimmt.
  - ✓ `einzahlen` **ist durch** `public` **für Klienten aufrufbar.**
- Die Realisierung dieser (zusammengehörigen) Dienstleistungen (als Methoden) ist verborgen.
  - ✓ **Kein Zugriff durch Klienten auf die Implementierung von** `einzahlen`
- Ebenso sind die Zustandsfelder als interne Strukturen eines Objekts gekapselt.
  - ✓ **Das Feld** `_saldo` **ist durch** `private` **vor externem Zugriff geschützt.**
- Auf den Zustand eines Objektes kann nur über seine Dienstleistungen zugegriffen werden.
  - ✓ **Hier durch** `einzahlen`



## Auswertung: Grobstruktur einer Klassendefinition

```
/**
 * Schnittstellenkommentar der Klasse
 */
class Girokonto
{
  private int _saldo;

  public void einzahlen( int betrag )
  {
      _saldo = _saldo + betrag;
  }
}
```

Kopf der Klasse

Rumpf der Klasse

**Klassenkopf**: spezifiziert den Namen der Klasse und beschreibt mit dem Schnittstellenkommentar die Aufgabe der Klasse.

**Klassenrumpf**: beinhaltet Zustandsfelder, Konstruktoren und Methoden, die die Zuständigkeiten der Klasse realisieren.



## Auswertung: allgemeine Struktur einer Klassendefinition

```
class Girokonto
{
  private int _saldo;

  public void einzahlen( int betrag )
  {
      _saldo = _saldo + betrag;
  }
}
```

Konstruktor kann fehlen (Standardkonstruktor)

class <*Klassenname*>
{
  <*Felder*>
  <*Konstruktoren*>
  <*Methoden*>
}



## Klassendefinition mit explizitem Konstruktor

```
class Girokonto
{
  private int _saldo;

  public Girokonto()
  {
      _saldo = 0;
  }

  public void einzahlen( int betrag )
  {
      _saldo = _saldo + betrag;
  }
}
```

Konstruktor kann explizit angegeben werden, heißt immer wie die Klasse

class *<Klassenname>*
{
  *<Felder>*
  *<Konstruktoren>*
  *<Methoden>*
}



## Objekte erzeugen

**Objekterzeugung**:
Objekte müssen zur Laufzeit durch einen expliziten Ausdruck erzeugt werden. Dazu wird ein eigenes **Schlüsselwort** (in Java `new`) verwendet. Wir nennen dies von nun an **Exemplarerzeugung** .

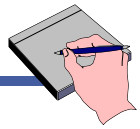
```
class Zeichner {
  ...
  Quadrat wand = new Quadrat();
  Dreieck dach = new Dreieck();
  Quadrat fenster = new Quadrat();
  ...
  wand.vertikalBewegen(80);
  fenster.farbeAendern("blau");
  dach.horizontalBewegen(70);
  ...
}
```

**Schlüsselwort**: Zeichenfolge, die in einer Programmiersprache eine feste Bedeutung hat (z.B. `class`). Schlüsselwörter sind (meist) reserviert, d.h. sie dürfen nicht als Namen von Variablen verwendet werden.



## Exemplarerzeugung und Konstruktor



- Eine **Exemplarerzeugung** (in Java mit `new`) bewirkt zweierlei:
  1. Ein **neues Objekt** der genannten Klasse wird erzeugt.
  2. Bei diesem Objekt wird der angegebene **Konstruktor ausgeführt**; ein Konstruktor **initialisiert** ein neu erzeugtes Objekt.

```
class Zeichner {
  ...
  Quadrat wand = new Quadrat();
  Dreieck dach = new Dreieck();
  Quadrat fenster = new Quadrat();
  ...
  wand.vertikalBewegen(80);
  fenster.farbeAendern("blau");
  dach.horizontalBewegen(70);
  ...
}
```

1

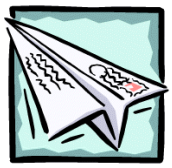
: Quadrat

2

```
class Quadrat {
  ...
  public Quadrat()
  {
    <Initialisierungen>
  }
  ...
}
```



## Methoden aufrufen



- Jeder **Methodenaufruf** richtet sich immer an ein bestimmtes Objekt, den **Adressaten** des Aufrufs.
- Der Adressat ist entweder explizit angegeben:

```
wand.vertikalBewegen(80);
```

  Die gerufene Methode ist dann üblicherweise Teil der **Schnittstelle** des gerufenen Objektes.
- Oder es wird eine Methode des aktuellen Objektes aufgerufen:

```
zeichneDach(80);
```

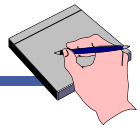
  **Hilfsmethoden**, die nur innerhalb einer Klasse verwendet werden, werden `private` deklariert.

**Botschaft**: Der Aufruf einer Methode wird oft auch als das Senden einer Botschaft oder Nachricht an das gerufene Objekt dargestellt. Dabei umfasst die Botschaft einen Bezeichner für das Objekt (als Adressaten), den Namen der Methode und die aktuellen Aufrufparameter.



## Die Punktnotation der Objektorientierung



- Die Methoden eines Objekts werden in vielen objektorientierten Sprachen in der **Punktnotation** (engl.: dot notation) aufgerufen.

```
wand.vertikalBewegen 80
```

Das aufzurufende **Objekt** wird benannt.

Der **Punkt-Operator** sagt aus, dass auf einen Bestandteil der Schnittstelle (also eine Dienstleistung des Objektes) zugegriffen werden soll.

Die aufzurufende **Methode** wird benannt, eventuell gefolgt von **aktuellen Parametern** (hier die ganze Zahl **80**).



## Die Punktnotation in Java

- Java folgt der objektorientierten Tradition und verwendet ebenfalls die Punktnotation für Methodenaufrufe an Objekten.

```
wand.vertikalBewegen(80);
```

Referenz auf ein Objekt

der Punkt

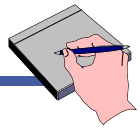
Die **aktuellen Parameter** werden in Java in **runden Klammern** eingeschlossen.

Auch wenn keine Parameter definiert sind, werden bei Methodenaufrufen die **runden Klammern immer** angegeben.



## Struktur der Methodendefinition in Java



**Methodenköpfe**: Klassen spezifizieren mit den Köpfen ihrer öffentlichen Methoden Dienstleistungen, die festlegen, wie die Zustände der Objekte sondiert oder verändert werden können. Die öffentlichen Methoden bilden die Schnittstelle einer Klasse.

**Methodenrümpfe**: realisieren die versprochenen Dienstleistungen durch eine Implementierung (konkrete Deklarationen und Anweisungsfolgen).

Die Unterscheidung zwischen der Schnittstelle (dem „Kopf“) und der Implementierung (dem „Rumpf“) einer Methode spiegelt sich auch in ihrer Struktur wider.

Hier ist in Java die **Signatur** enthalten.

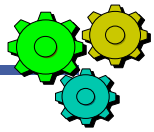
```
/**
 * Kommentar, der die Funktionalität der Methode beschreibt
 */
<Zugriffsmodifikator> <Ergebnistyp> <Name> ( <Parameter> )
{
        <(imperative) Anweisungen>
}
```

Kopf der Methode

Rumpf der Methode



## Verändernde Methoden



- Pragmatik: Wenn Methoden den Zustand ihres Objektes verändern (**verändernde Methoden**, engl.: mutators), dann sollten sie keinen Wert zurück geben.
- Für Klienten sind nur die Methoden aufrufbar, die mit `public` als „öffentlich“ deklariert wurden; sie bilden die Schnittstelle einer Klasse.
- Neben den öffentlichen Methoden werden zur Implementierung oft interne Methoden verwendet. Sie werden in Java als `private` deklariert und entsprechen dem ursprünglichen imperativen Prozedurbegriff.

für Klienten sichtbar

benutzerdefinierter Bezeichner für die Methode (Pragmatik für Java: klein geschrieben)

```
public void einzahlen( int betrag )
```

Parametername

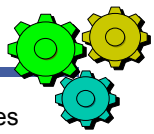
Methode liefert keinen Rückgabewert

formaler Parameter vom Typ `int`

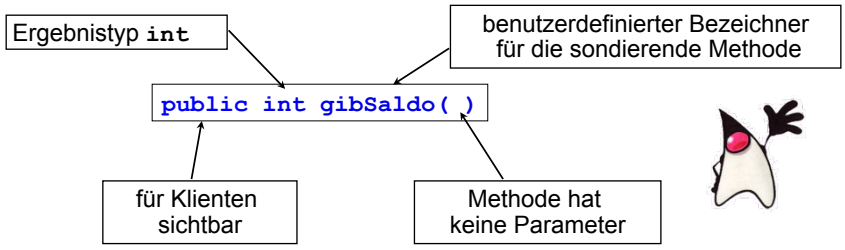


## Sondierende Methoden



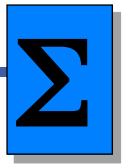
- **Sondierende Methoden** (engl.: accessor methods) sollen den Zustand des Objektes, an dem sie gerufen werden, nicht verändern.
- Sondierende Methoden liefern einen (Ergebnis-) Wert von einem vereinbarten (Ergebnis-) Typ.
- Das Ergebnis wird explizit (mittels der **`return`** Anweisung) zurückgegeben.
- Solche Methoden können deshalb an der Aufrufstelle als Teil von Ausdrücken verwendet werden.








## Zusammenfassung



- **Klassendefinitionen** beschreiben Klassen.
- Wir erzeugen Objekte durch **Exemplarerzeugungen**., bei denen immer ein Konstruktoren aufgerufen wird.
- Ein Konstruktor **initialisiert** den Zustand eines Objektes.
- Die (Zustands-)**Felder** eines Objektes halten seinen Zustand; in einer Klassendefinition bezeichnen wir die Definitionen der Felder auch als **Exemplarvariablen**.
- Eine Methode besteht aus einem **Kopf** und einem **Rumpf**.
- Wir unterscheiden **sondierende** (nur lesende) Methoden und **verändernde** Methoden.

