Objekte und Klassen

Objekte, Klassen, Felder und Methoden, Ergebnistyp, Parameter, Signatur, Kommentare, Javadoc

Algorithmen

- Algorithmen sind Handlungsanweisungen
  - beispielsweise um ein Essen zu kochen
  - um ein Auto zu bauen
  - um sich an der Uni einzuschreiben
  - um die jährliche Tilgung eines Kredits zu berechnen
- Elementare Aktionen werden als bekannt vorausgesetzt
  - rühren, erhitzen, Gemüse beigeben, …
  - Schraube eindrehen, Punkt schweißen, …
  - Formular ausfüllen, Geld überweisen, …
  - Multiplizieren, addieren, …
- Ein Algorithmus beschreibt eindeutig
  - Reihenfolge
  - Bedingungen

der Ausführung dieser elementaren Aktionen

Programme

- Programme sind formale Beschreibungen von Algorithmen
- Programme werden in einer der Aufgabe gemäßen Sprache formuliert

Spaghettiprogramm:

Man nehme
- 2 l Wasser
- 250 g Spaghetti

1. Koche Wasser
2. Gebe Spaghetti bei
3. Solange (Spaghetti nicht al dente)
   Koche Spaghetti

Tilgungsprogramm:

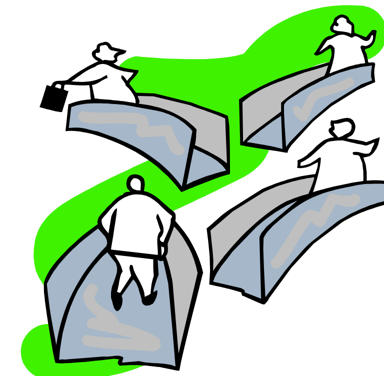
Sei
- K der Kredit
- z der Zinssatz
- T die Tilgung
- N das Jahr

1. Setze N=0
2. Solange K > 0
   a. N = N+1
   b. K = K*(1+z/100) – t
3. Drucke N

©H. Peter Gumm, Philipps-Universität Marburg

Programme müssen alles können

- Ein Programm sagt einem Rechner alles was er zu tun hat
  - öffne Editor
  - setze Cursor
  - nehme Tastatur-Eingabe entgegen
  - spiele einen Klang



- Dabei müssen alle Eventualitäten berücksichtigt werden
  - Ist die Maus bewegt worden ?
  - Muss der Uhrzeiger weiter gestellt werden ?
  - Wurde eine Sondertaste gedrückt ? Welche ?

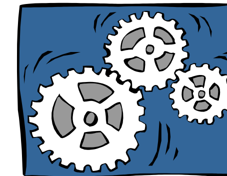
©H. Peter Gumm, Philipps-Universität Marburg

Programme sind komplex

- Programme werden immer größer, komplexer und unüberschaubarer
  - Wo ist der letzte Tastendruck gespeichert ?
  - Wo ist der Programmteil, der für die Maus zuständig ist ?
- Nach einer Weile versteht niemand mehr das Programm
  - Den Programmteil, der für die Eingabe zuständig ist hat der Kollege geschrieben, und der ist bei der Konkurrenz
  - Was ist denn das hier für ein Trick – besser nicht anfassen !
- Es kann nach einiger Zeit nicht mehr geändert werden
  - Die Cobol-Programmierer der deutschen Banken sind längst in Rente
  - Wer kann heute noch Cobol

So war es früher. Heute hat man ..

- Objekte
  - Objekte sind kleine Programmstücke.
  - Jedes Objekt hat spezifische Fähigkeiten.
  - Objekte kooperieren, um eine umfangreiche Aufgabe gemeinsam zu erfüllen.



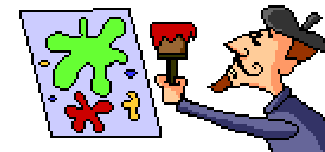
- Klassen
  - Klassen sind Fabriken für Objekte.
  - Jede Klasse kann einen ganz bestimmten Typ von Objekten erzeugen.
  - Jedes Objekt gehört zu genau einer Klasse.

... statt eines fetten Programms

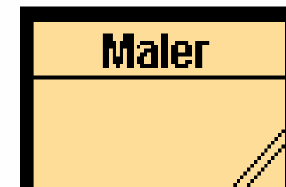
©H. Peter Gumm, Philipps-Universität Marburg

Objekte und Klassen

- Objekte modellieren
  - Gegenstände
  - Akteure
- Objekte haben
  - Eigenschaften (Felder)
  - Methoden (Fähigkeiten)
- Klassen
  - Umfassen gleichartige Objekte
  - Enthalten den Bauplan für Objekte
  - Erzeugen Objekte



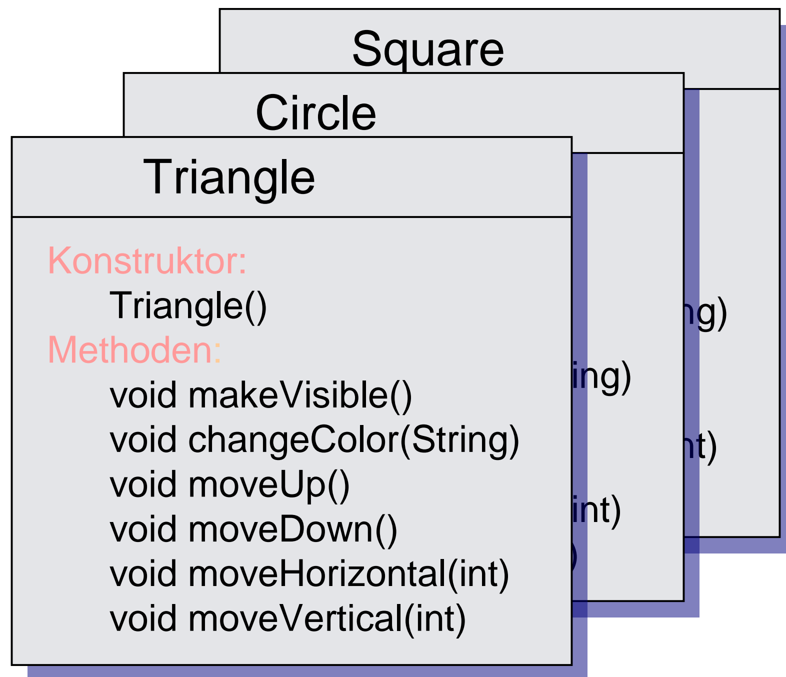
geerbt von Object
void maleHaus()
void malePortrait()
void säuberePinsel()
void ändereFarbe()

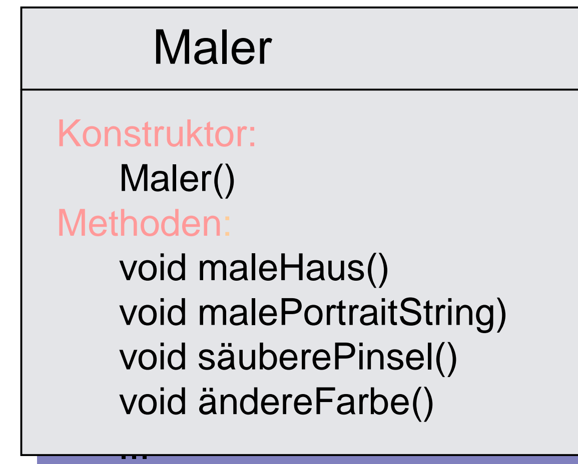


©H. Peter Gumm, Philipps-Universität Marburg

Karteikarten

- Auf Karteikarten notieren wir die Fähigkeiten der Objekte





void: das Ergebnis der Methode ist nicht ein Wert, sondern eine Änderung des Zustandes

Anhand dieser Karteikarten kann sich ein Programmierer passende Objekte aussuchen und diese in seinen Programmen verwenden

©H. Peter Gumm, Philipps-Universität Marburg

Bankkonten

Wir wollen Konten als Objekte modellieren

- Was ist ein Konto ?
  - Es hat eine Nummer
  - Es kann den Namen des Inhabers speichern
  - Es kann einen Geldbetrag speichern
- Was kann man mit Konten machen ?
  - Den Kontostand abfragen
  - Einen Geldbetrag einzahlen
  - Einen Geldbetrag abheben
  - Auf ein anderes Konto einen Betrag überweisen.

konto_1

nummer : 621 736 172

inhaber : “Donald Duck“

kontoStand : 100 000 000

getKontoStand

einzahlen **betrag :**

abheben **betrag :**

überweisen **empfänger :** **betrag :**

©H. Peter Gumm, Philipps-Universität Marburg

Modellierung als Klasse

Wir konzentrieren uns zunächst auf die ***Objektfelder***

nummer
inhaber
kontoStand

- Die Klasse `Konto` dient als „*Fabrik*" bzw. als „*Vorlage*" für beliebig viele einzelne Konten
- Jedes Konto ist ein Objekt dieser Klasse. Die Objektfelder speichern jeweils eigene Werte.

```
class Konto
{
      int nummer;
      String inhaber;
      int kontoStand;
}
```

Einige Objekte der Klasse Konto:

konto_1	
nummer :	12334
inhaber :	"Otto"
kontoStand :	700

konto_2	
nummer :	76622
inhaber :	"Bill"
kontoStand :	19000

deinKonto	
nummer :	89871
inhaber :	"Ötzi"
kontoStand :	-230

©H. Peter Gumm, Philipps-Universität Marburg

In BlueJ

- In BlueJ erzeugen wir die Klasse **Konto**
  - Neue Klasse: **Konto**
  - Objektfelder:
    - nummer
    - inhaber
    - kontoStand

Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen... | V

```
/** Beschreibung der Klasse Konto.
 *
 * @author H.P.Gumm
 * @version 6.9.06
 */
public class Konto {
//====================================
//   Objekt-Felder
//====================================
    int nummer;
    String inhaber;
    int kontoStand;

//====================================
//   Konstruktoren
```

Klasse übersetzt - keine Syntaxfehler

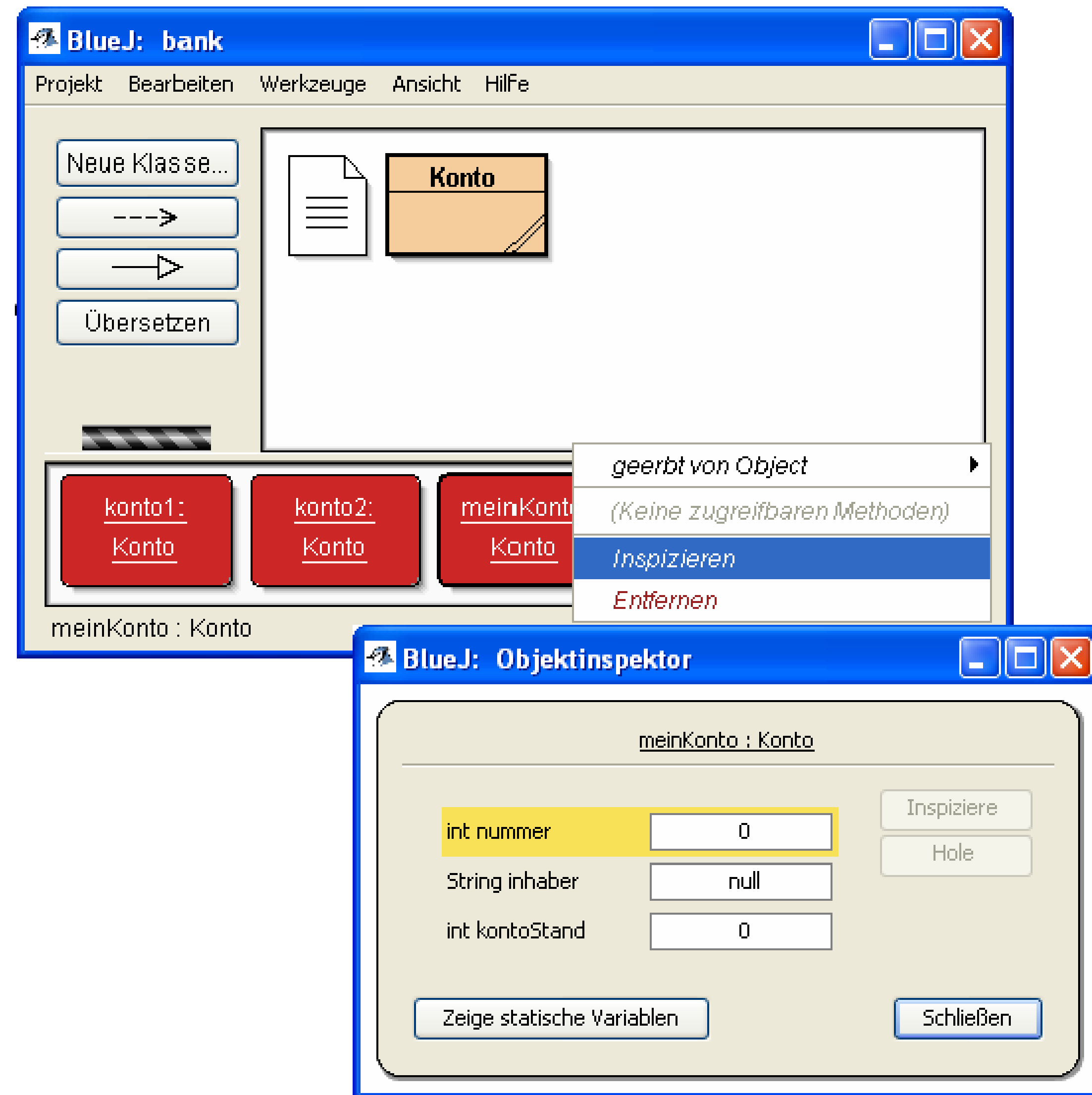
gespeichert

©H. Peter Gumm, Philipps-Universität Marburg

Ein Objekt der Klasse Konto

Dann erzeugen wir Objekte

- `konto1`, `konto2` und `meinKonto`
- Wir inspizieren `meinKonto` und finden
  - `int nummer:      0`
  - `String inhaber: null`
  - `int kontoStand: 0`
- `null` bedeutet: *uninitialisiert*



©H. Peter Gumm, Philipps-Universität Marburg

Typen von Feldern

- Die Klasse Konto hat drei Objektfelder
  - `nummer`
  - `inhaber`
  - `kontoStand`

```
class Konto
{
      int nummer;
      String inhaber;
      int kontoStand;
}
```

- Jedem Feld ist sein **Typ** vorangestellt:
  - `int nummer ;`
    - das Feld `nummer` kann eine beliebige ***ganze Zahl***)* aufnehmen, keinen Text, keine Dezimalzahl …
  - `String inhaber ;`
    - das Feld `inhaber` kann einen beliebigen ***Text*** aufnehmen, keine Zahl, keinen Wahrheitswert, …
  - `int kontoStand ;`
    - das Feld `kontoStand` kann eine beliebige ***ganze Zahl***)* aufnehmen, keinen Text, keine Dezimalzahl …

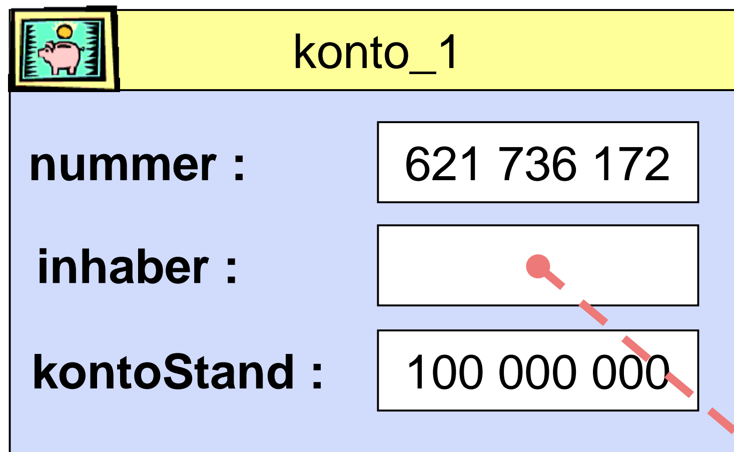
*) genaugenommen, eine Zahl zwischen -2199023255552 und +2199023255551

©H. Peter Gumm, Philipps-Universität Marburg

Primitive Typen - Objekttypen

Es gibt zwei Kategorien von Typen :

- `int` ist ein *primitiver Typ*
  - Der Wert eines primitiven Typs wird direkt in das Feld geschrieben
- die anderen primitive Typen heißen:
  - `boolean, char, byte, short, long, float, double`
- `String` ist ein *Objekttyp* – so wie ein `Konto` oder `Maler`.
  - In ein Feld vom Objekttyp wird nicht sein Wert, sondern ein Link (Referenz) auf den Wert hineingeschrieben.
  - Solange es noch nicht initialisiert ist, hat es den „Wert“ `null`



Im Hauptspeicher des Rechners:

“Hallo Leute“ “Asterix“ “Die spinnen, die Römer“ “Donald Duck“ “Was ist los?“

©H. Peter Gumm, Philipps-Universität Marburg

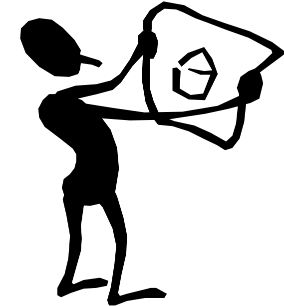
Methoden

- Bisher haben Konten nur Felder
  - nummer
  - inhaber
  - kontoStand
- Es fehlen noch Methoden
  - getKontoStand
  - einzahlen
  - abheben

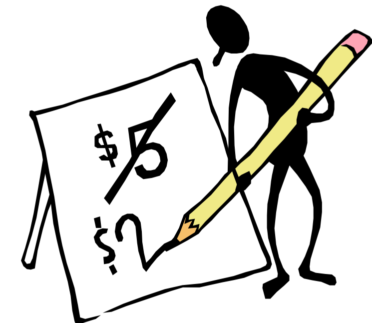
©H. Peter Gumm, Philipps-Universität Marburg

Observer - Mutator

Methoden lassen sich klassifizieren



- *Observer*-Methoden liefern Objekt-Informationen
  - Sie *liefern* einen Wert zurück
    - z.B. eine Zahl, einen String, ein anderes Objekt, …
  - Observer sollen das Objekt nicht verändern



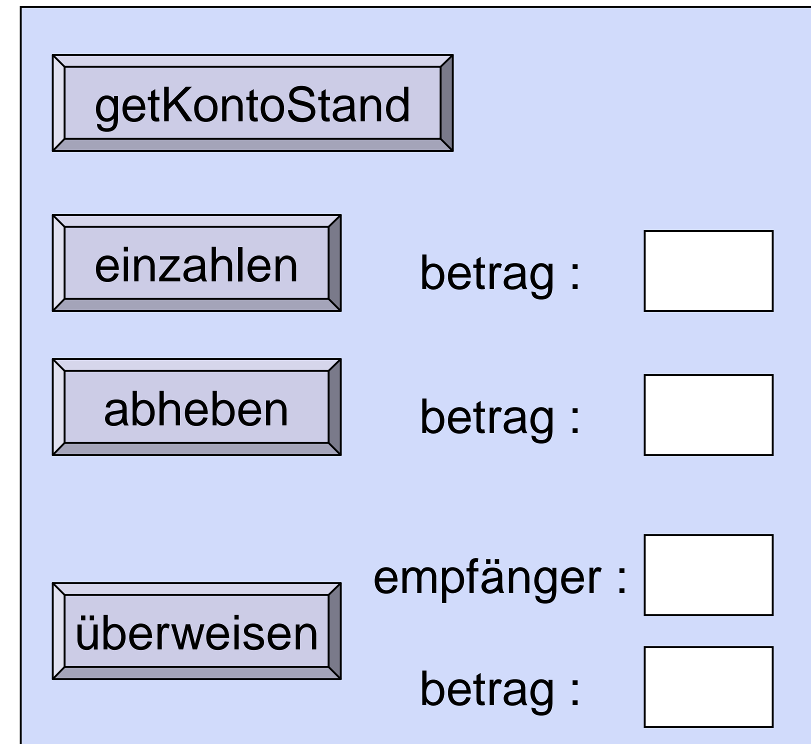
- *Mutator*-Methoden verändern das Objekt
  - Sie verändern den Zustand des Objekts
  - Sie werden i.A. nicht verwendet, um Informationen über das Objekt zu liefern

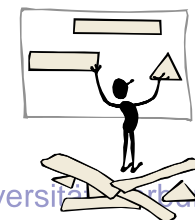
- Methoden können Mutator und Observer sein
  - Nur in begründeten Ausnahmen zu verwenden
  - … ansonsten berechnet der Tutor Ihnen Minuspunkte

©H. Peter Gumm, Philipps-Universität Marburg

Observer – Mutator für das Konto

- Observer
  - `getKontoStand`
  - Typ des Rückgabewertes: `int`
- Mutatoren
  - `einzahlen`
  - `abheben`
  - `überweisen`
  - Typ des Rückgabewertes : keiner, also `void`





©H. Peter Gumm, Philipps-Universität Marburg

Methoden mit Resultat

getKontoStand

- Die Methode `getKontostand` liefert den Wert des Feldes `kontoStand`

Typ des Ergebnisses | Name der Methode | Parameter-liste

```
int getKontoStand( )
{
      return kontoStand;
}
```

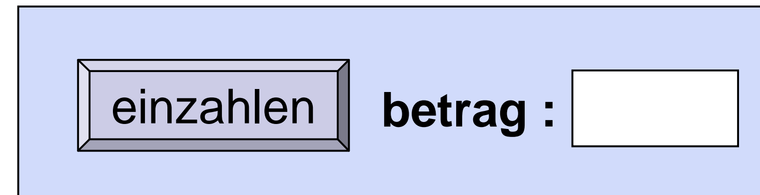
liefere als Resultat …

Den Inhalt des Feldes **kontoStand**

©H. Peter Gumm, Philipps-Universität Marburg

Methoden mit Parameter

- Die Methode `einzahlen` benötigt einen Input:
  - Den zu überweisenden `betrag`.
  - Das ist ein Wert vom Typ `int`.



Typ des Ergebnisses | Name der Methode | ein Parameter vom Typ int

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

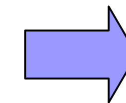
©H. Peter Gumm, Philipps-Universität Marburg

Signatur

- Den Kopf einer Methode nennt man auch *Signatur*
  - Kennzeichnend für die Signatur sind nur
    - der **Name**
    - die Liste der **Parametertypen**

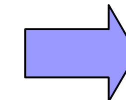
- Ergebnistyp ist offiziell nicht Teil der Signatur.
  - Parameternamen auch nicht

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



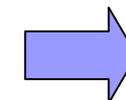
einzahlen(int)

```
int getKontoStand( )
{
        return kontoStand;
}
```



getKontoStand( )

```
void überweisen( Konto empfänger, int betrag )
{
        // ... kommt noch  ;
}
```



überweisen(Konto, int)

©H. Peter Gumm, Philipps-Universität Marburg

Die neue Klasse Konto

Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen...

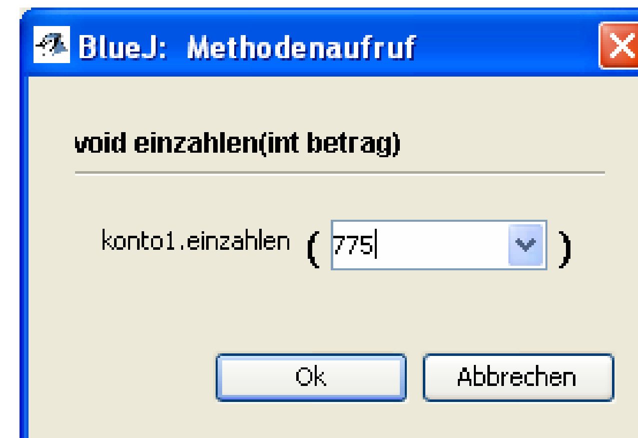
```
public class Konto {
//===================================
//    Objekt-Felder
//===================================
    int nummer;
    String inhaber;
    int kontoStand;
//===================================
//    Objekt-Methoden
//===================================
    int getKontoStand(){
        return kontoStand;
    }

    void einzahlen(int betrag){
        kontoStand += betrag;
    }
}// Ende der Klasse Konto
```

- Beachten Sie die Formatierung der Klammern für den Rumpf der Methoden
- Das ist üblich
- Wenn der Cursor neben einer Klammer steht, wird deren Partner hervorgehoben

©H. Peter Gumm, Philipps-Universität Marburg

Die neuen Fähigkeiten von Konto

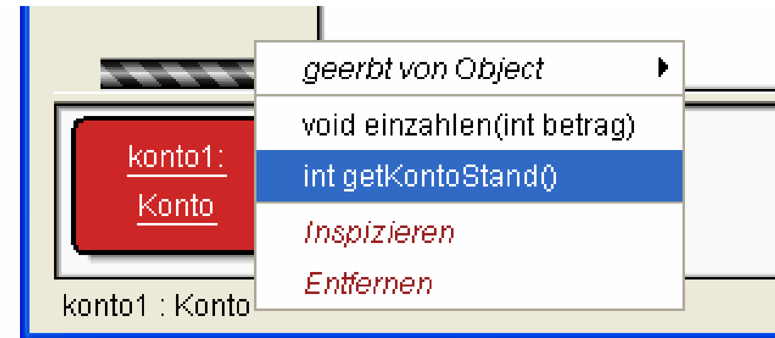
- Das Objekt versteht jetzt **einzahlen** und **getKontoStand**
- Wir wählen diesen Menüpunkt.
- Es erscheint ein Fenster um den Parameter einzugeben:

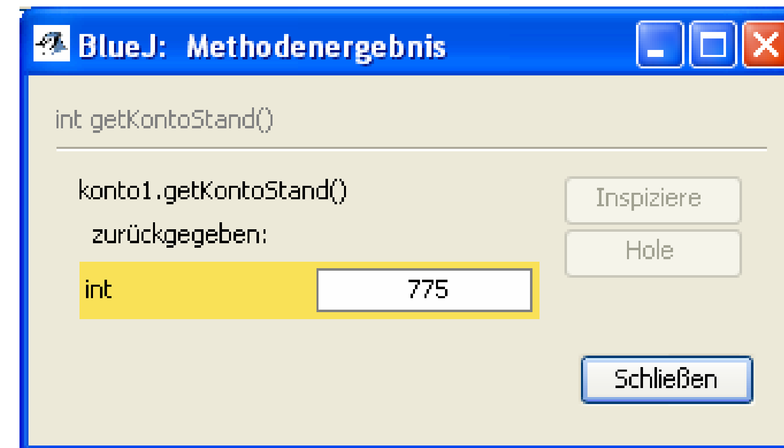


©H. Peter Gumm, Philipps-Universität Marburg

Rückgabewerte

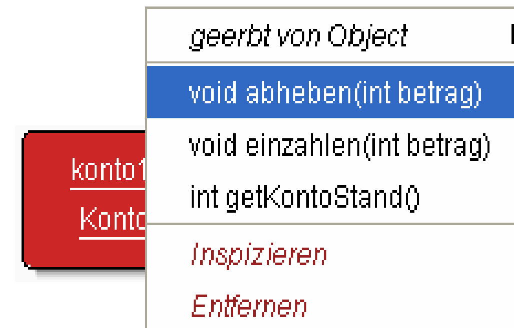
- **getKontoStand()** ist ein Observer
- der Rückgabewert ist vom Typ int
- BlueJ öffnet ein Fenster, in dem dieser angezeigt wird





©H. Peter Gumm, Philipps-Universität Marburg

Objekte „live“ beobachten



- Inspizieren Sie ein Konto
- Lassen Sie den Inspektor geöffnet und zahlen Sie ein, heben Sie ab, …
- Sie sehen, wie sich der Wert `kontoStand` dabei verändert
- Übrigens
  - wurde „`abheben`“ analog zu „`einzahlen`“ implementiert.
  - Statt “`+=`“ (*erhöhen um*) müssen Sie “`-=`“ (*erniedrigen um*) verwenden.

©H. Peter Gumm, Philipps-Universität Marburg

Überweisen

überweisen empfänger : betrag :

- Um einen Geldbetrag von einem Konto auf ein anderes zu überweisen, benötigt die Methode zwei Parameter
  - das Konto des Empfängers.
  - den Betrag

Typ des Ergebnisses | Name der Methode | Erster Parameter ist vom Typ `Konto` | Zweiter Parameter ist vom Typ `int`

```
void überweisen( Konto empfänger, int betrag )
{
      abheben(betrag);
      empfänger.einzahlen(betrag);
}
```

Konto des Empfängers | Aufruf der Methode | betrag

©H. Peter Gumm, Philipps-Universität Marburg

Methodenaufruf

- Beim *Aufruf* einer Methode werden formale Parameter durch konkrete Werte ersetzt

```
void überweisen( Konto empfänger, int betrag )
{
      abheben(betrag);
      empfänger.einzahlen(betrag);
}
```

Die Methode `überweisen` besteht aus
- § einem Aufruf der Methode `abheben,`
- § dann aus einem Aufruf der Methode `einzahlen` des `empfänger`-Kontos.

©H. Peter Gumm, Philipps-Universität Marburg

Kommunikation zwischen Objekten

Der Aufruf

```
überweisen(konto7,325);
```

führt zu

```
abheben(325);
konto7.einzahlen(325);
```

also einem Methodenaufruf eines anderen Objektes.

- Es liegt eine (elementare) Objekt-Kommunikation vor:
  - Das Objekt, das die Methode `überweisen` benutzt, kommuniziert mit dem Objekt, das als `empfänger`-Konto benannt ist.
  - Der `empfänger`, hier `konto7`, erhält die Weisung, `einzahlen` aufzurufen und den Parameter `325`.

©H. Peter Gumm, Philipps-Universität Marburg

Was passiert ?

meinKonto

nummer :	621 736 172
inhaber :	“Donald Duck“
kontoStand :	775

- getKontoStand
- einzahlen — betrag :
- abheben — betrag :
- überweisen — empfänger : (→ konto7), betrag : 325

konto7

nummer :	90908787
inhaber :	“Dagobert“
kontoStand :	100 000 000

- getKontoStand
- einzahlen — **betrag :**
- abheben — **betrag :**
- überweisen — **empfänger :**, **betrag :**

```
abheben(325);
```

≅ abheben — **betrag :** 325

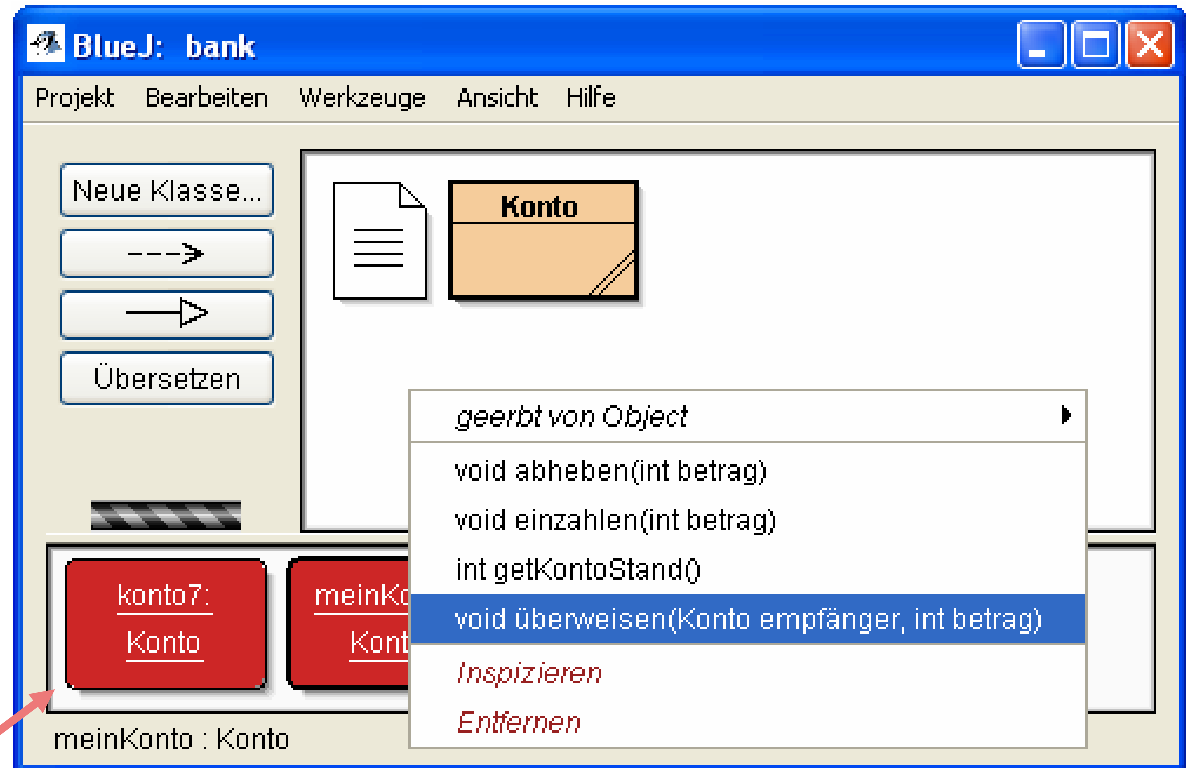
```
konto7.einzahlen(325);
```

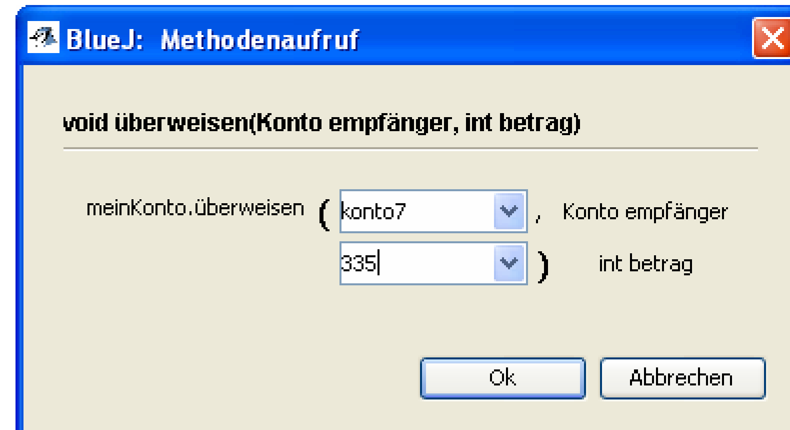
≅ konto7: einzahlen — **betrag :** 325

©H. Peter Gumm, Philipps-Universität Marburg

In BlueJ

- Eingabe des Konto-Parameters
  - durch Eingabe des Namens hier: konto7 (ohne “ “), oder...
  - durch Klick auf das Objekt





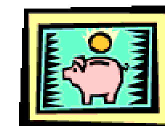
©H. Peter Gumm, Philipps-Universität Marburg

Konstruktoren

Konstruktoren dienen dazu, Objekte zu erzeugen

- Bisher haben alle Konten
  - Nummer: 0
  - Inhaber : <null>
- Nummer und Namen sollten *bei der Erzeugung* des Objektes gesetzt werden
  - Danach brauchen sie nicht mehr verändert werden.
- Wer sollte also verantwortlich sein für die Konstruktor-Methode
  - die Klasse ?
  - das Objekt ?

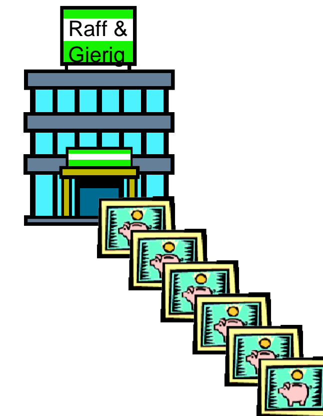
oder



?

Konstruktoren gehören zur Klasse.

- Sie haben den gleichen Namen wie die Klasse
  - Der Konstruktor der Klasse `Konto` heißt `Konto`
  - Groß- bzw. Kleinschreibung ist signifikant
    - Konto ≠ KOnto ≠ konto



- Konstruktoren haben keinen Rückgebewert
  - Ihr Ergebnis ist ein neues Objekt der Klasse
    - Ein Konstruktor für `Konto` liefert ein Objekt der Klasse `Konto`
- Sie brauchen kein `return`
  - Wenn ein Konstruktor fertig ist, hat er sein Objekt konstruiert
- Für jede Klasse gibt es einen Default-Konstruktor
  - Er kann ein Objekt erzeugen
  - Alle Felder sind mit Vorgabe-Wert initialisiert
    - Jedes int-Feld mit 0
    - Jedes Objekt-Feld mit <null>

Default-Konstruktor der Klasse Konto:

```
Konto(   )
{
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Ein besserer Konstruktor für Konten

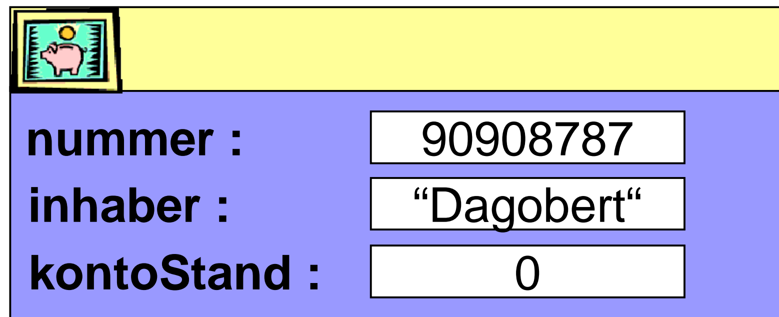
```
Konto(int neueNummer, String name){
      nummer   = neueNummer;
      inhaber = name ;
      kontoStand = 0;
}
```

Definition des Konstruktors

Der Aufruf

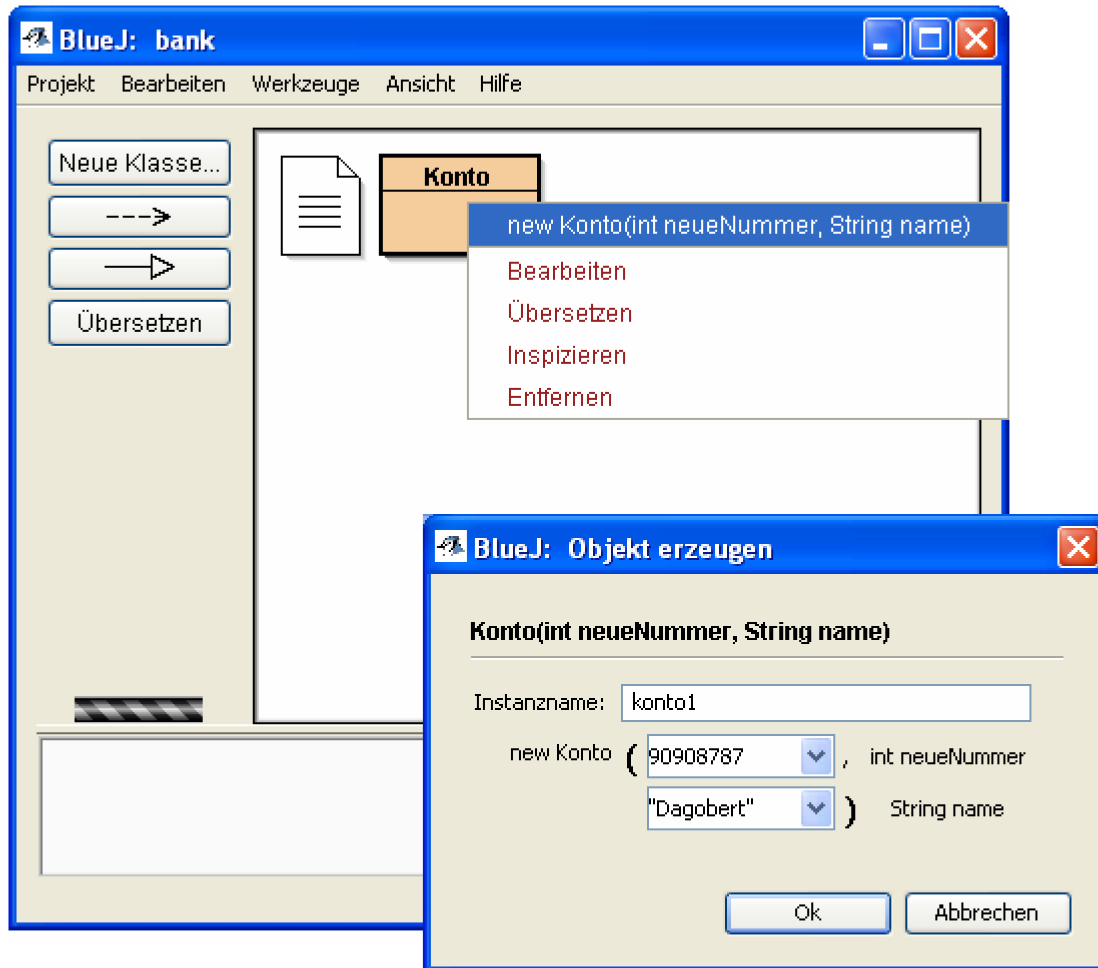
```
new Konto(90908787, “Dagobert“)
```

liefert:



©H. Peter Gumm, Philipps-Universität Marburg

Konto-Erzeugung in BlueJ



Die Klasse Konto hat einen neuen Konstruktor mit zwei Parametern

Bei der Erzeugung des Objektes werden die gewünschten Werte der Parameter abgefragt

©H. Peter Gumm, Philipps-Universität Marburg

Mehrere Konstruktoren

- Eine Klasse kann mehrere Konstruktoren haben
  - Beispiele:
    - Die Bank will dem Neukunden gleich ein Begrüßungsgeld überweisen
    - Wenn der Neukunde von einem anderen Kunden geworben wurde, soll der Werber eine Prämie erhalten

- Verschiedene Konstruktoren müssen verschiedene Signaturen haben
  - In der Klasse Konto sind z.B. gleichzeitig möglich
    - Konto ( ) → Konto ()
    - Konto (int nummer, String name) → Konto (int, String)
    - Konto (String name, int nummer) → Konto (String, int)
    - Konto (int nummer, String name, int begrüßungsGeld) → Konto (int, String, int)
    - Konto (int nummer, String name, Konto werber) → Konto (int, String, Konto)
  - Nicht erlaubt sind gleichzeitig
    - Konto (int nummer, String name) → Konto (int, String)
    - Konto (int betrag, String empfänger) → Konto (int, String)

Signatur :

Nicht alles was erlaubt ist, ist auch sinnvoll.

©H. Peter Gumm, Philipps-Universität Marburg

Information hiding

- Bisher gehört zu einem Konto die folgende Indexkarte:

Konto

Felder:
- int nummer
- String inhaber
- int kontoStand

Konstruktoren:
- Konto()
- Konto(int,String)

Methoden:
- int getKontoStand ()
- void einzahlen (int)
- void abhebeben (int)
- void überweisen (Konto, int)
- ...

Ein Benutzer der Klasse Konto sollte auf die gespeicherten Daten nur mit Hilfe der Methoden Zugreifen können.

Wir schützen die Felder vor externem Zugriff durch das Schlüsselwort
private

Was sind die Vorteile,
was die Nachteile ?

©H. Peter Gumm, Philipps-Universität Marburg

Daten schützen

- Wir schützen die Felder **inhaber** und **kontoStand**.
- Von einem Benutzer können sie dann nicht gelesen oder verändert werden, außer mit den erlaubten Methoden
  - **int getKontoStand()**
  - **String getInhaber()**
  - **einzahlen()**
  - **abheben()**
  - **überweisen()**
- Sollte die Bank später z.B.
  - Transaktionsgebühren einführen
  - Protokolle schreiben
  - Auszüge erstellen

  - dann muss sie nur die obigen Methoden ändern
  - und kann sicher sein, dass alle Transaktionen erfasst wurden.
- Auch im täglichen Leben
  - kann der Bankangestellte kein Geld vom Konto nehmen, ohne ein Formular auszufüllen
  - darf er nicht den Namen des Inhabers ändern, es sei denn mit einer festgelegten Prozedur/Methode

Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen Rückgängig Ausschneiden Kopieren Einfügen Suchen...

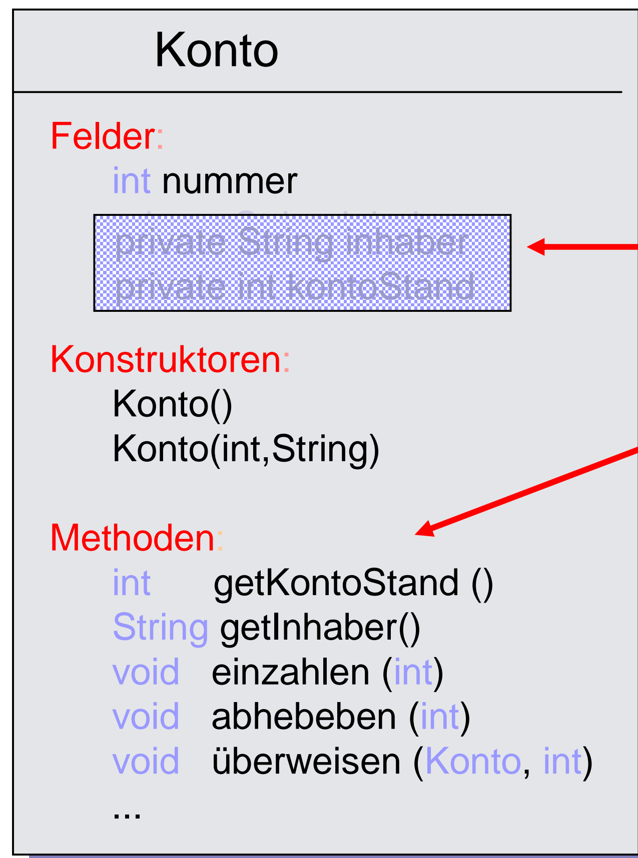
```
//=================================
//   Objekt-Felder
//=================================
    int nummer;
    private String inhaber;
    private int kontoStand;

//=================================
//   Objekt-Methoden
//=================================
    String getInhaber(){
       return inhaber;
    }

    int getKontoStand(){
        return kontoStand;
    }

    void einzahlen(int betrag){
```

geändert

©H. Peter Gumm, Philipps-Universität Marburg

Die Karteikarte für `Konto`

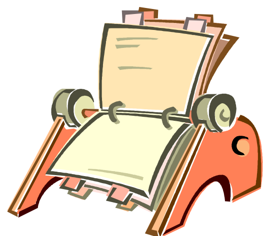
- Die Karteikarte für Konto sieht jetzt so aus:



`inhaber` und **`kontoStand`** sind von außerhalb nicht mehr sichtbar

`inhaber` kann noch gelesen, aber nicht verändert werden

will die Bank später für jedes Lesen eine Legitimation (z.B. Passwort) verlangen, so muss sie nur **`getInhaber()`** ändern.



©H. Peter Gumm, Philipps-Universität Marburg

Klassendesign

- Daten der Klassenobjekte in Feldern gespeichert
  - Felder nach außen nicht zugreifbar
- Zugriffe nur über kontrollierte Methoden
  - getter – zum Lesen
  - setter – zum Schreiben
- Vorteile
  - kontrolliertes Verändern
    - setMonat(2) funktioniert nur,
      falls getTag() <= 28,
      oder falls getTag() = 29 und istSchaltJahr()
  - Veränderbarkeit
    - Neue Version von Date kann Datum anders speichern
      - z.B. als Anzahl der Tage seit 1.1.1970
      - alle getter und setter Methoden bleiben
    - Programme, die nur die getter und setter benutzen, können unverändert die neue Version verwenden.

```
Datum
--------------------------------
Felder:
    private int tag
    private int monat
    private int jahr
    private boolean schaltJahr

Konstruktoren:
    Datum()
    Datum( int, int, int)

Methoden:
  getter
    public int getTag()
    public int getMonat()
    public int getJahr()
    public boolean istSchaltJahr()
      …
  setter
    public void  setTag(int)
    public void  setMonat(int)
    public void  setJahr(int)
    public void  addDays(int)
      …
```


©H. Peter Gumm, Philipps-Universität Marburg

Vorteile

- kontrolliertes Verändern
  - setMonat(2) funktioniert nur, falls getTag() <= 28, oder falls getTag() = 29 und istSchaltJahr()
- Veränderbarkeit
  - Neue Version von Date kann Datum anders speichern
    - z.B. als Anzahl der Tage seit 1.1.1970
    - alle getter und setter Methoden bleiben
  - Programme, die nur die getter und setter benutzen, können unverändert die neue Version verwenden.

```
Datum
--------------------------------
Felder:
      private int tageSeit1970

Konstruktoren:
      Datum()
      Datum( int, int, int)

Methoden:
  getter
      public int getTag()
      public int getMonat()
      public int getJahr()
      public boolean istSchaltJahr()
         ...
  setter
      public void  setTag(int)
      public void  setMonat(int)
      public void  setJahr(int)
      public void  addDays(int)
         ...
```


©H. Peter Gumm, Philipps-Universität Marburg

Kommentare

- Java hat drei Sorten von Kommentaren
  - // Kommentar bis zum Zeilenende

    // kurze Erläuterung für jemand, der das Programm liest
  - /* Alles in diesen Klammern */
    - erläutert eine schwierige Stelle
    - damit ich später weiß was ich mir dabei gedacht habe
    - damit mein Kollege den Code versteht
    - damit mein Tutor versteht, was ich gemacht habe
      - schließlich will ich die volle Punktzahl bekomme
  - /** Aus dem Text in diesen Klammern */
    - erzeugt `javadoc.exe` eine Dokumentation
    - Bestimmte Marken zur Textauszeichnung erlaubt, z.B.:
      - @author
      - @date
      - @param
      - @return
    - Die API-Dokumentation ist auf diese Weise erzeugt

©H. Peter Gumm, Philipps-Universität Marburg

Javadoc sollte wissen:

- Zu Beginn der Klasse:
  - /** Wozu dient sie, was stellt sie bereit
  - @author, @version */
- Für jedes Feld
  - /** Was wird hier gespeichert */
- Für jede Methode
  - /** Was tut sie */
  - @param, ggf.: @return

©H. Peter Gumm, Philipps-Universität Marburg

Mindest-Dokumentation

Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen... | Weitersuchen | Schließen | Implementierung

```
/** Basisklasse für Konten.
 * @author H.P.Gumm
 * @version 6.9.06
 */
public class Konto {
//=========================================================
//   KlassenFelder
//=========================================================
    /** Default Zinssatz
     *  Kann in Unterklassen geändert werden */
     static int aktuellerZinssatz = 3;

    /** Nächste zu vergebende Nummer */
     static int kontenAnzahl = 0;
//=========================================================
//   Objekt-Felder
//=========================================================
    /** Kontonummer */
    static int nummer;

    /** aktuelle Kontostand */
    int kontoStand;
```

Klasse übersetzt - keine Syntaxfehler

gespeichert

- Dokumentation ist essentiell
  - Andere müssen meine Klassen nutzen, ohne den Source-Code zu kennen
- Dokumentation wird direkt aus BlueJ erzeugt
  - Ändere **Implementierung** zu **Schnittstelle**
- Mindest-Dokumentation
  - Wozu dient die Klasse
    - Autor
    - Version/Datum
  - Was bedeuten die Felder
    - nicht notwendig für als *private* erklärte
  - Was tun die Methoden
- Dokumentation wird direkt aus BlueJ erzeugt

Ergebnis: Brauchbare Dokumentation

Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen...

Class Konto

```
java.lang.Object
  └ Konto
```

```
public class Kontoextends java.lang.Object
```

Basisklasse für Konten.

Version:
6.9.06

Author:
H.P.Gumm

Field Summary

(package private) static int	**aktuellerZinssatz** Default Zinssatz Kann in Ur geändert werden
(package private) java.lang.String	**inhaber** Nachname des Inhabers
(package private) static int	**kontenAnzahl** Nächste zu vergebende Nu

Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen... | Weitersuchen | Schlie

Method Summary

(package private) void	**abheben**(int betrag)
(package private) void	**einzahlen**(int betrag) Einzahlen auf aktuelles Konto
(package private) java.lang.String	**getInhaber**() Liefert Namen des Kontoinhabers
(package private) int	**getKontoStand**()
(package private) void	**überweisen**(Konto empfänger, int betrag) Überweisung auf ein anderes Konto

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

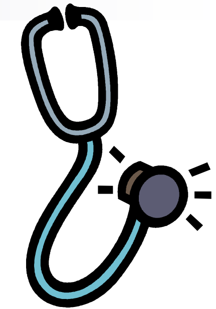
aktuellerZinssatz

```
static int aktuellerZinssatz
```

Default Zinssatz Kann in Unterklassen geändert werden

©H. Peter Gumm, Philipps-Universität Marburg

Weitere javadoc Möglichkeiten



Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen... | Weitersuchen | Schließen | Implementierung

```
/** Überweisung auf ein anderes Konto
 *  @param empfänger das Konto des Empfängers
 *  @param betrag der zu überweisende Betrag
 *  <p><b>Negativer Betrag führt zu einer Exception</b>
 */
void überweisen(Konto empfänger, i
    if(betrag>0) {
        abheben(betrag);
        empfänger.einzahlen(betrag
    else throw new Exception("Halt
}
```

Klasse übersetzt - keine Syntaxfehler

Konto

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen... | Weitersuchen

überweisen

```
void überweisen(Konto empfänger,
                int betrag)
         throws java.lang.Exception
```

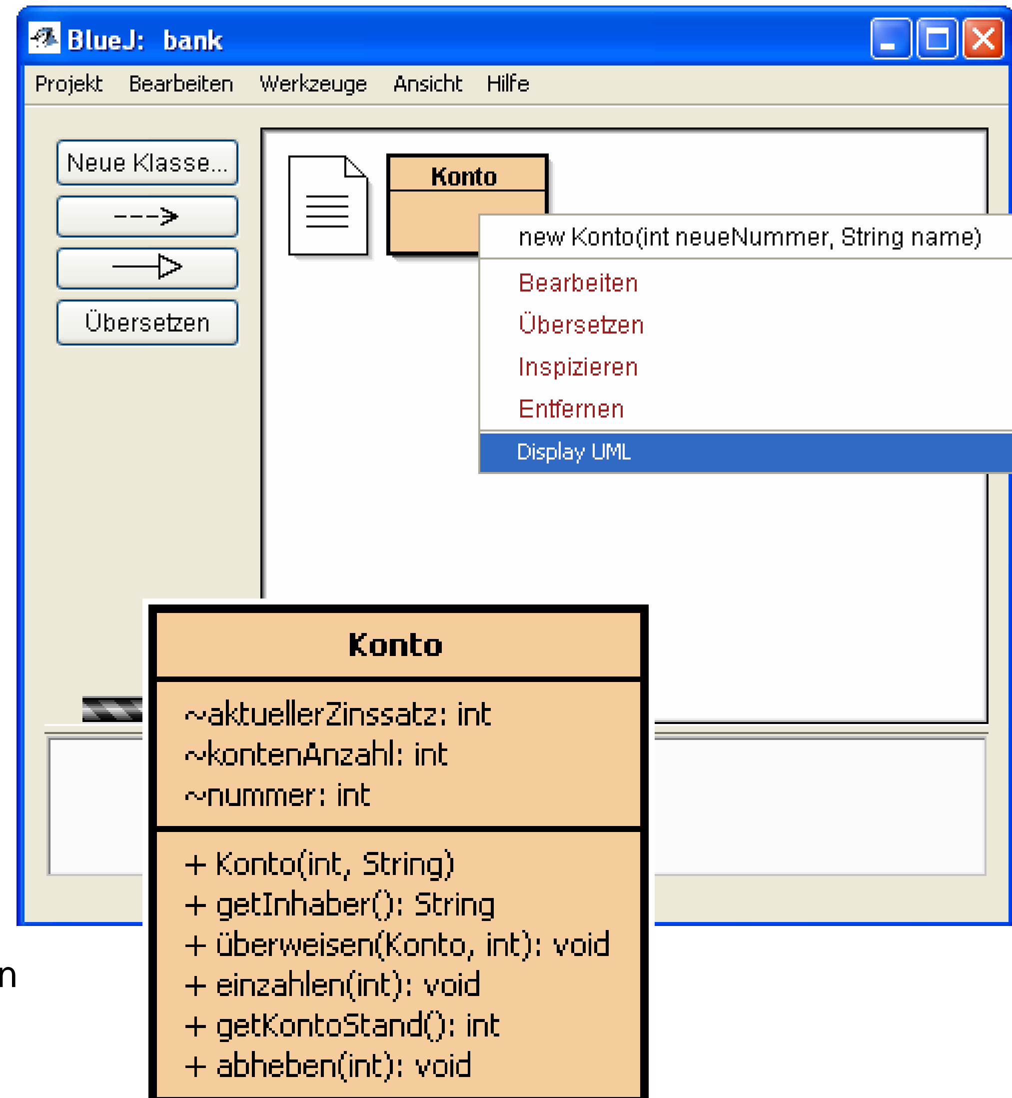
Überweisung auf ein anderes Konto

Parameters:
`empfänger` - das Konto des Empfängers
`betrag` - der zu überweisende Betrag
Negativer Betrag führt zu einer Exception
Throws:
`java.lang.Exception`

- Neben Schlüsselwörtern wie
  - @param
  - @return
- auch einfache HTML-Befehle möglich
  - <p>, <b>, <strong>, <i>, ...

UML-Diagramme

- UML = Unified Modelling Language
  - Sammlung von Diagrammtypen
  - Keine Sprache (language)
- Klassendiagramme = Karteikarten
  - zeigt Felder und Methoden
  - nur falls nicht private
- Format
  - Präfix
  - Signatur
  - Resultattyp
- Präfixe
  - \+ falls public
  - ~ falls kein Modifier
- Als BlueJ-Extension verfügbar
  - UMLextension.jar herunterladen
  - in `/lib/extensions` kopieren
  - zusätzlicher Eintrag im Klassenmenü



©H. Peter Gumm, Philipps-Universität Marburg

Ein Hauptprogramm

- Java-Anwendungen werden mit einer speziellen Methode gestartet
  - **main**
  - fest vorgeschriebene Signatur
- In dieser Methode können
  - Variablen deklariert und initialisiert
  - Methoden aufgerufen werden

RaffUndGierig

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen | Rückgängig | Ausschneiden | Kopieren | Einfügen | Suchen... | Weitersuche

```
public class RaffUndGierig {

  public static void main(String[] args){

      // ...
      // Anweisungen des Hauptprogranmmes
      // ...
  }// Ende von main

}// Ende der Klasse RaffUndGierig
```

geändert

©H. Peter Gumm, Philipps-Universität Marburg

main

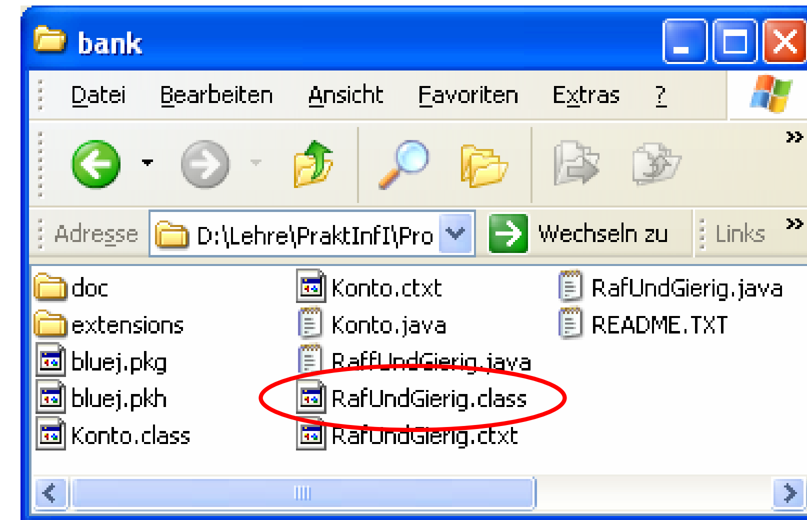
- Zunächst werden drei Variablen vom Typ Konto eingeführt und initialisiert
- Dann wird Geld eingezahlt und kreuz und quer überwiesen
- Dann werden Kontoauszüge und Bilanz gedruckt

```
/** Implementiert das Bankhaus RaffUndGierig.
 *  Hat nur drei Kunden und muss bald wieder schliessen.
 * @author H.P.Gumm
 * @version 6.9.06
 */
public class RaffUndGierig {

  public static void main(String[] args){
  // Einige Kontos werden eröffnet
    Konto k1 = new Konto(1001,"Anton");
    Konto k2 = new Konto(1002,"Bill");
    Konto k3 = new Konto(1003,"Chris");

  // Geld wird eingezahlt und verschoben
    k1.einzahlen(300);
    k2.einzahlen(200);
    k2.überweisen(k3,100);

  // Anton überweist alles was er hat an Chris
    k1.überweisen(k3,k1.getKontoStand());

  // Die Kontoauszüge
    System.out.println(
      k1.getInhaber()+" hat "+k1.getKontoStand()+"\n"+
      k2.getInhaber()+" hat "+k2.getKontoStand()+"\n"+
      k3.getInhaber()+" hat "+k3.getKontoStand()+"\n");

  // Bilanz der Bank
      System.out.println("Summe der Einlagen = "+
                              (k1.getKontoStand()+
                               k2.getKontoStand()+
                               k3.getKontoStand()));
  } // Ende von main
```


©H. Peter Gumm, Philipps-Universität Marburg

Starten des Hauptprogramms

- Ohne BlueJ
  - Aufruf der kompilierten Klasse, die main enthält
    - hier: **`RaffUndGierig.class`**
  - mittels **`java.exe`**
  - von der Konsole
  - **`-classpath`** sagt java, wo sich die benötigten Klassen befinden



C:\WINDOWS\system32\cmd.exe

der Aufruf

das Ergebnis

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Dokumente und Einstellungen\Peter>cd ../..

C:\>java -classpath D:\Lehre\PraktInfI\Programme\bank RaffUndGierig
Anton hat 0
Bill hat 100
Chris hat 400

Summe der Einlagen = 500

C:\>
```


©H. Peter Gumm, Philipps-Universität Marburg

mit BlueJ

- mit BlueJ
  - rechte Maustaste
  - main rufen
  - Ok
  - Ergebnis erscheint in Konsolenfenster

©H. Peter Gumm, Philipps-Universität Marburg