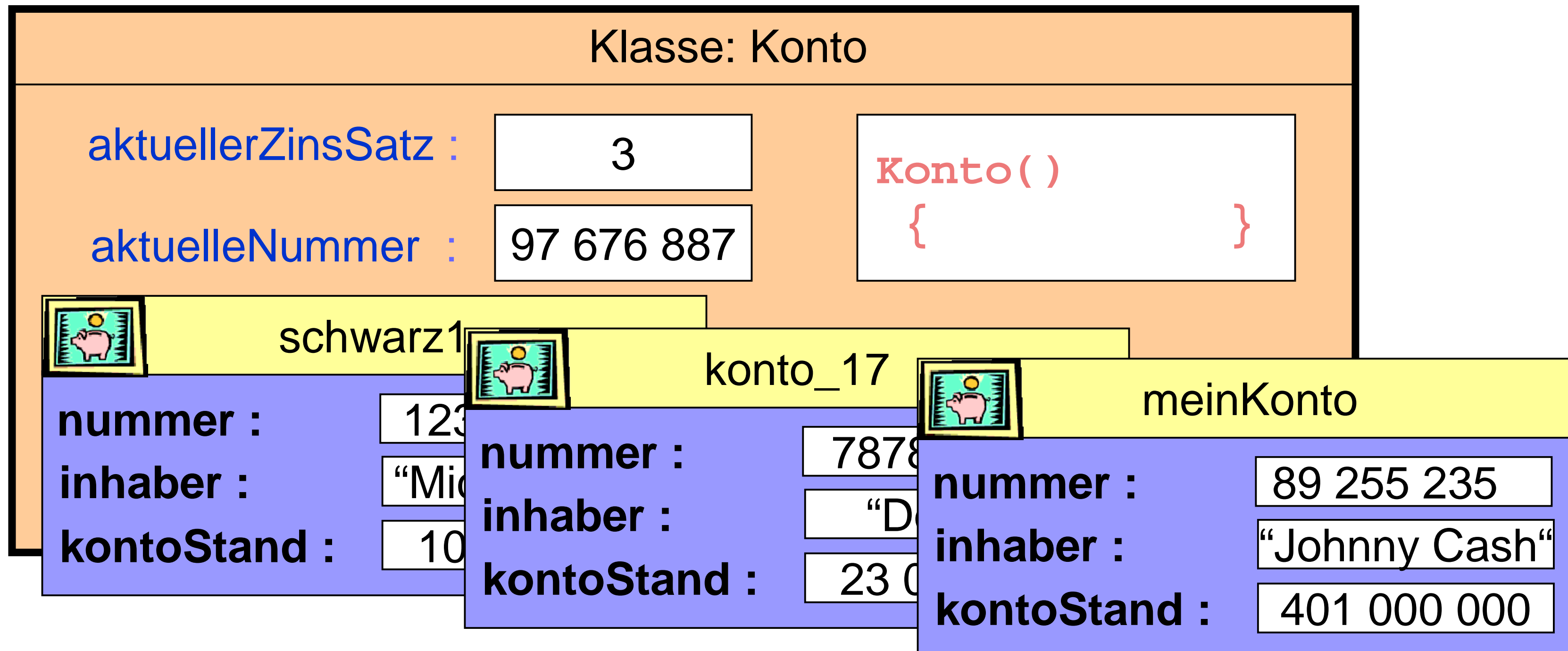
Klassen und Unterklassen

Klassenfelder, Klassenmethoden, Applikationen,Unterklassen, Benutzung, Vererbung, super, over-riding, Systemklassen, Pakete, Applets

Klassenfelder

- Manche Felder müssen nicht in jedem Objekt gespeichert werden, es reicht, wenn sie einmal in der Klasse vorhanden sind
  - der aktuelle Zinssatz
  - die nächste zu vergebende Kontonummer
- Klassenfelder sind jedem Objekt der Klasse jederzeit zugänglich
  - das Objekt kann den aktuellen Zinssatz erfragen
- Klassenfelder heißen in Java: static



©H. Peter Gumm, Philipps-Universität Marburg

Statische Felder

```
class Konto{
// Klassenfelder
     static int aktuellerZinsSatz = 3 ;
     static int aktuelleKontoNummer;

// Objektfelder
     int nummer;
     String inhaber = "Der Herr Direktor";
     int kontoStand = 0 ;
```

- Klassenfelder werden durch das Schlüsselwort static gekennzeichnet
- Jede Zeile, die mit "//" beginnt, ist ein Kommentar
- Klassenfelder und Objektfelder können initialisiert werden

©H. Peter Gumm, Philipps-Universität Marburg

Autoinkrement : x++

```
class Konto{
// Klassenfelder
   static int aktuellerZinsSatz = 3 ;
   static int aktuelleKontoNummer;

// Konstruktor
    Konto(String neuerKunde){
        inhaber = neuerKunde;
        kontoStand = 0;
        nummer = aktuelleKontoNummer;
        aktuelleKontoNummer++;
    }

// Objektfelder
    int nummer;
    String inhaber;
    int kontoStand;

// Objektmethoden
...
```

Für jedes int-Feld **xyz** bewirkt die Anweisung

xyz++ ;

dass der Inhalt von **xyz** um 1 erhöht wird.

Hier wird automatisch die nächste freie Kontonummer vergeben

Erhöhe die aktuelle Kontonummer um 1

©H. Peter Gumm, Philipps-Universität Marburg

Klassen-Methoden = static Methoden

```
class Konto{
  static zinsSatz = 3;

  static int getZinsSatz(){
    return zinsSatz();
  }
}
```

- Klassen-Methoden sind nützlich um
  - statische Felder zu manipulieren
    - z.B. den Zinssatz zu ändern
  - Observatoren zu implementieren, die unabhängig von dem aufrufenden Objekt sind
    - z.B. printAGB()   // drucke das „Kleingedruckte“
    - z.B. getZinssatz()
  - Konstruktoren kennen wir schon als Klassen-Methoden
    - Weder benötigen Sie das Schlüsselwort static, noch eine explizite Angabe des Rückgabewertes
    - Die Klasse produziert ihre Objekte

©H. Peter Gumm, Philipps-Universität Marburg

Methoden aufrufen

- Wenn wir eine Objekt-Methode nutzen, so müssen wir zunächst Objekte der Klasse erzeugen und dann deren Methode aufrufen:

```
Square wand   = new Square();
Circle sonne = new Circle();

sonne.changeColor("yellow");
wand.moveHorizontal(50);
```

Objekte
- sonne,
- wand

- Wenn wir eine Klassen-Methode (static method) aufrufen wollen, so können wir statt eines Objekts die Klasse angeben.

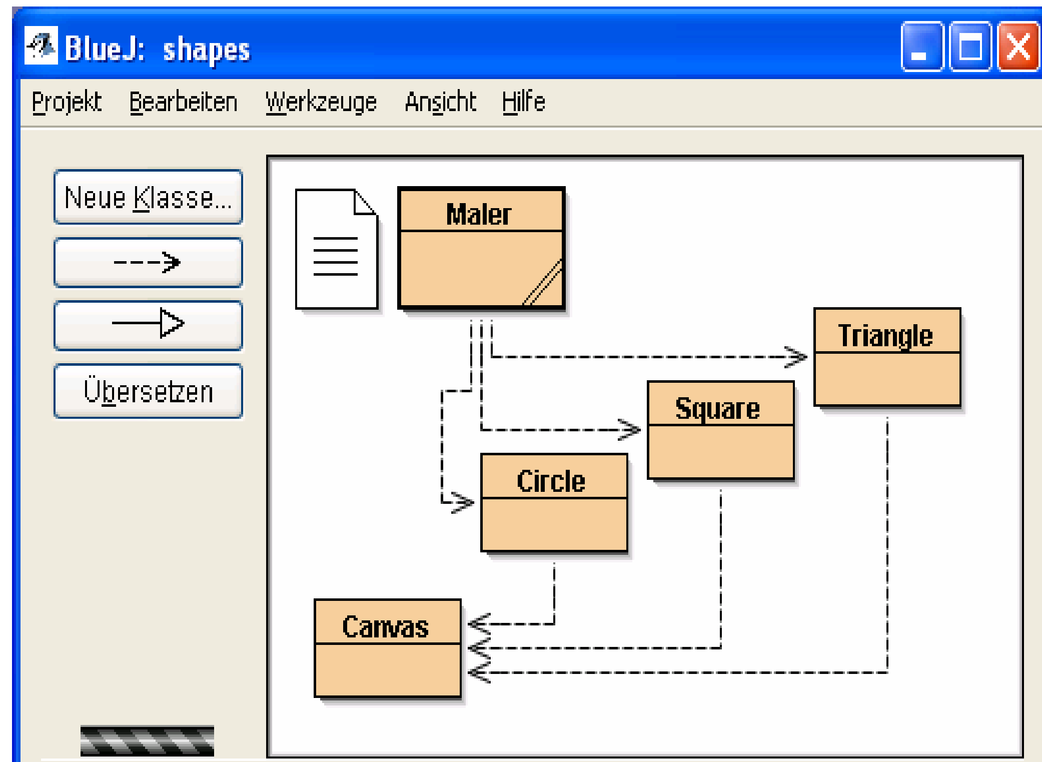
```
SparKonto.getZinsSatz();
```

Klasse SparKonto

©H. Peter Gumm, Philipps-Universität Marburg

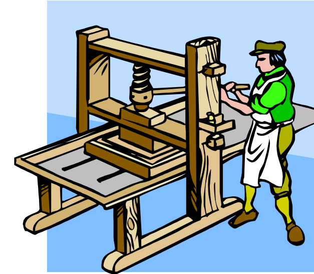
Vorhandene Klassen *benutzen*

- Die Klasse **`Maler`** hat vorhandene Klassen **`Circle`**, **`Square`** und **`Triangle`** *benutzt*.
  - **`Circle`**, **`Square`** und **`Triangle`** haben die Klasse **`Canvas`** *benutzt*.
  - Wir konnten diese Klassen nutzen, ohne sie im Detail zu verstehen
- In BlueJ wird die *benutzt*-Relation durch ge-strichelte Pfeile dargestellt



©H. Peter Gumm, Philipps-Universität Marburg

Eingebaute Klasse: System



- **System**
  - Eine Klasse mit systemnahen Feldern und Methoden
  - Wichtig ist vor allem das statische Feld **out**.
    - Es enthält ein „*Schreibobjekt*“ (einen sog. „**PrintStream**“).
    - **PrintStreams** kennen die Methode **println**(), mit der man im Terminalfenster schreiben kann.

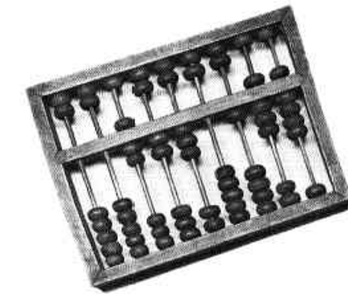
```
System.out.println(“Ey, Leute, was‘n los“ );
System.out.println(17+4);
System.out.println(meinKonto); // Was kommt da wohl raus ?
```

oder auch: (mithilfe des Pakets java.io)

```
PrintStream ausgabe = System.out;
ausgabe.println(“Is ja bald vorbei“ );
ausgabe.println(“17 + 4 = “ + (17+4)); //Raten Sie mal
```


©H. Peter Gumm, Philipps-Universität Marburg

Eingebaute Klasse: Math



- **Math**
  - Eine Klasse mit einer Sammlung von
    - mathematischen Funktionen
      - z.B. **max**, **min**, **abs**, **random**, **sin**, **log**, ...
    - und mathematischen Konstanten
      - **PI**, **E**
  - **Math** hat nur statische Methoden und Felder
    - Man braucht also keine Math-Objekte zu erzeugen

```
System.out.println(Math.max(3,5));
System.out.println(Math.abs(x*x-10*x+1));

System.out.println(Math.sqrt(Math.PI + Math.E));
System.out.println(Math.random());
```


©H. Peter Gumm, Philipps-Universität Marburg

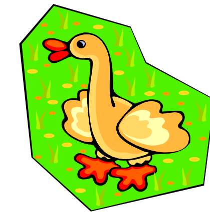
Die Klasse String

- `String`
  - die Objekte sind Zeichenketten,
    - genaugenommen: Listen von Zeichen.
  - Strings kann man
    - verketten (vornehm: konkatenieren)
    - schreiben
    - z.B. mit `System.out.println( );`
  - Es gibt Methoden um
    - die Länge festzustellen: `length( )`
    - zwei Strings auf Gleichheit zu testen: `equals( )`
    - Das i-te Zeichen zu bestimmen: `charAt( )`
    - …

©H. Peter Gumm, Philipps-Universität Marburg

Spezielle Syntax für String

- String-Erzeugung
  - Angabe als Zeichenfolge in Gänsefüßchen
    - `"Otto", "Hallo Welt", "Gruss, Dein \n Julius"`
  - im Programm mit Konstruktor
    - `String gruss = new String("Halli hallo");`
  - oder direkt
    - `String gruss = "Halli hallo";`

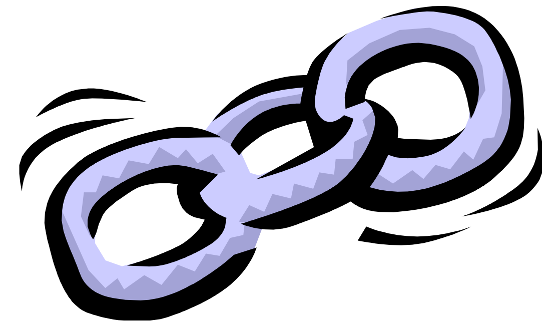


- Konkatenation mit „+“
  - `"Hallo" + "  Welt"` ergibt: `"Hallo  Welt"`
  - `"Bitte" + "nicht" + "stören"` ergibt: `"Bittenichtstören"`
- Alle Java-Typen haben String-Darstellung
  - Umwandlung automatisch von links nach rechts:
  - `System.out.println("1 und 1 ist "+(1+1));`
  - `System.out.println("1 und 1 ist "+1+1);`

©H. Peter Gumm, Philipps-Universität Marburg

Zeichenketten

- Java-Datentyp String
  - Literale
    - In `" "` eingeschlossene Folgen von Zeichen
  - Operationen
    - `+` (Verkettung, Konkatenation)
      - `"Hallo " + " Welt"`
    - Methode: `length( )`
      - `"Hallo".length()`
  - Strings sind Objekte !
    - Vergleich mit == unzuverlässig
      - Besser equals() verwenden
        - `"Hallo".equals("Hallo")`
  - Jedes Java-Objekt erbt die Methode `toString()`



©H. Peter Gumm, Philipps-Universität Marburg

Applikationen

- Eine Applikation ist eine *stand-alone* Java-Anwendung
- Start von der Kommandozeile:
  - `java HauptKlasse`

- Eventuell mit Parametern P_0, P_1, ..., P_k

- Diese können im Programm gelesen werden, z.B. als
  - String P_0 = args[0];
  - String P_1 = args[1];
  - ...

C:\WINDOWS\System32\cmd.exe

```
C:\Beispiel>java HauptKlasse Eins 2 Drei 4
Gruss aus der HilfsKlasse
Argumente sind
Eins 2 Drei 4
C:\Beispiel>
```


©H. Peter Gumm, Philipps-Universität Marburg

Bestandteile einer Applikation

- Eine Applikation besteht aus
  - mindestens einer public deklarierten Haupt-Klasse
    - z.B.: **HauptKlasse**
  - in der sich eine Methode **main**
  - mit der Signatur
    public static void main(String[] args)

  befinden muss.

- **args** ist hier ein Array von **String**, der zur Laufzeit die Kommandozeilen-Parameter enthält.

```
public class HauptKlasse
{
     public static void main(String[] args){
       HilfsKlasse.sagWasNettes();
       System.out.println("Argumente sind");
       for (int k=0; k<args.length; k++)
            System.out.print(args[k]+" ");
     }
}
```

Voraussetzungen für den Aufruf

- Alle benutzten Klassen liegen compiliert in vor
  - z.B.: `HauptKlasse.class, HilfsKlasse.class`
- HauptKlasse muss eine main-Methode haben:
  - `public static void main(String[] etwas)`
- `java.exe` ist im Suchpfad (PATH)
- alle benutzten Klassen sind im Klassenpfad (CLASSPATH)

C:\WINDOWS\System32\cmd.exe

```
C:\>cd Beispiel

C:\Beispiel>dir
 Volume in drive C is SYSTEM
 Volume Serial Number is 2C73-6EA0

 Directory of C:\Beispiel

10/10/2002  01:42 PM    <DIR>          .
10/10/2002  01:42 PM    <DIR>          ..
10/10/2002  01:17 PM               827 HauptKlasse.class
10/10/2002  01:16 PM               486 HilfsKlasse.class
               2 File(s)          1,313 bytes
               2 Dir(s)   3,282,186,240 bytes free

C:\Beispiel>java HauptKlasse Eins 2 Drei 4
Gruss aus der HilfsKlasse
Argumente sind
Eins 2 Drei 4
C:\Beispiel>_
```


©H. Peter Gumm, Philipps-Universität Marburg

Gepackte Verzeichnisse

```
C:\WINDOWS\System32\cmd.exe
C:\>cd Beispiel

C:\Beispiel>dir
 Volume in drive C is SYSTEM
 Volume Serial Number is 2C73-6EA0

 Directory of C:\Beispiel

10/10/2002  02:03 PM    <DIR>          .
10/10/2002  02:03 PM    <DIR>          ..
10/10/2002  01:17 PM             3,051 GrossProjekt.jar
10/10/2002  01:17 PM               827 HauptKlasse.clas
10/10/2002  01:16 PM               486 HilfsKlasse.clas
               3 File(s)          4,364 bytes
               2 Dir(s)   3,282,132,088 bytes free

C:\Beispiel>java -jar GrossProjekt.jar alpha beta gamma
Gruss aus der HilfsKlasse
Argumente sind
alpha beta gamma
C:\Beispiel>
```

- Man kann das komplette Verzeichnis in ein komprimiertes Archiv packen, z.B. `GrossProjekt.jar`
- Endung: `.jar` (für java archive)
- jar-Files können auch mit *WinZip* ausgepackt werden
- Sie können unausgepackt ausgeführt werden:
  - `java -jar HauptKlasse.jar`

©H. Peter Gumm, Philipps-Universität Marburg

jar-Dateien aus BlueJ

- Klassen compilieren und **main** testen
- *Project/ Als Verzeichnis speichern*
- Klasse mit **main** auswählen
- Quelltext mitspeichern
- Archivnamen wählen, z.B. meinProgramm

©H. Peter Gumm, Philipps-Universität Marburg

Unterklassen

- Wir wollen spezielle Konten implementieren
  - **Girokonten**
    - Man kann überziehen, muss dann aber Zinsen zahlen
    - Es gibt keine Habenzinsen
  - **Sparkonten**
    - man erhält Zinsen,
    - man kann nicht überziehen
    - man kann von Sparkonten nicht überweisen
- Sparkonten und Girokonten sind spezielle Konten
  - Jedes `GiroKonto` *ist* auch ein Konto
    - Die Klasse `GiroKonto` ist eine ***Unterklasse*** der Klasse `Konto`
  - Jedes `SparKonto` *ist* auch ein Konto
    - Die Klasse `SparKonto` ist eine ***Unterklasse*** der Klasse `Konto`

©H. Peter Gumm, Philipps-Universität Marburg

Vererbung / Erweiterung

- Unterklassen und ihre Objekte haben zusätzliche Eigenschaften
  - Sie „erben“ alle (nicht privaten) Felder und Methoden der Vaterklasse
    - Sowohl Sparkonten als auch Girokonten haben automatisch
      - Die Felder nummer, kontoStand, inhaber
      - Die Methoden getKontoStand, einzahlen, abheben
  - Sie können zusätzliche Felder haben
    - Sparkonten
      - Was ist der sparZinsSatz
      - Wann wurde letztmals Zins gutgeschrieben
    - Girokonten
      - Was ist der Überziehungszins
      - Was ist das Überziehungslimit
      - Seit wann ist überzogen
  - Sie können zusätzliche Methoden haben
    - Sparkonten
      - Zinsen gutschreiben
    - Girokonten
      - Gebühr abziehen

©H. Peter Gumm, Philipps-Universität Marburg

Unterklassen in Java

- Mit dem Schlüsselwort **extends** beerbt eine Klasse eine andere Klasse:

```
class SparKonto extends Konto{
    ...
  static int sparZinsSatz = 3;
  int letzterZinsTag;

  int getSparZinsSatz(){
     return sparZinsSatz;
  }

  void zahleZinsen(){
        . . .
```

```
class GiroKonto extends Konto{
       ...
  static int kreditZinsSatz = 13;
  int letzterRechnungsTag;
  int limit=10000;

  int getLimit(){
       return limit;
  }

  void berechneGebühr(){
        . . .
```


©H. Peter Gumm, Philipps-Universität Marburg

super

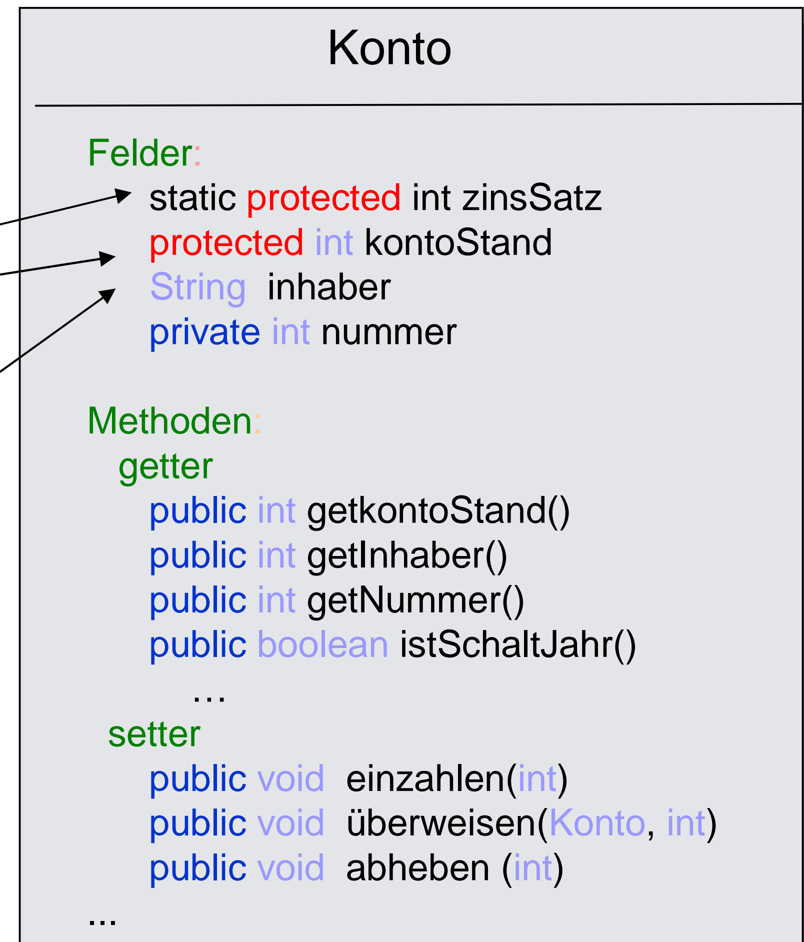
- verweist von der Unter- in die Oberklasse

```
class SparKonto extends Konto{

  // Wir übernehmen den
  // Konstruktor der Oberklasse

  SparKonto(String x){
       super(x);
       letzterZinsTag = 0;
  }
       ...
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Zugriff auf Oberklasse

- private Felder sind privat
  - auch aus der Unterklasse aus nicht zugreifbar
  - getter und setter sollten genügen
- protected Felder sind
  - zugreifbar aus der Unterklasse
  - nicht zugreifbar aus anderen Klassen
- ungeschützte Felder sind
  - zugreifbar von allen Klassen des gleichen Pakets
  - nicht zugreifbar aus Klassen anderer Pakete
  - „package private“



©H. Peter Gumm, Philipps-Universität Marburg

Zugriffe aus der Unterklasse

Konto

Felder:
- protected static int zinsSatz
- protected int kontoStand
- String inhaber
- private int nummer

Methoden:
- getter
  - public int getkontoStand()
  - public int getInhaber()
  - public int getNummer()
  - public boolean istSchaltJahr()
  - …
- setter
  - public void einzahlen(int)
  - public void überweisen(Konto, int)
  - public void abheben (int)

...

```
class SparKonto extends Konto{

  void zahleZinsen(){
    einzahlen( kontoStand * zinsSatz);
  }  // o.k.

  void druckeAuszug(){
   System.out.println(inhaber); //o.k.
   System.out.println(nummer); //falsch
   System.out.println(kontoStand); //o.k.
  }
```


©H. Peter Gumm, Philipps-Universität Marburg

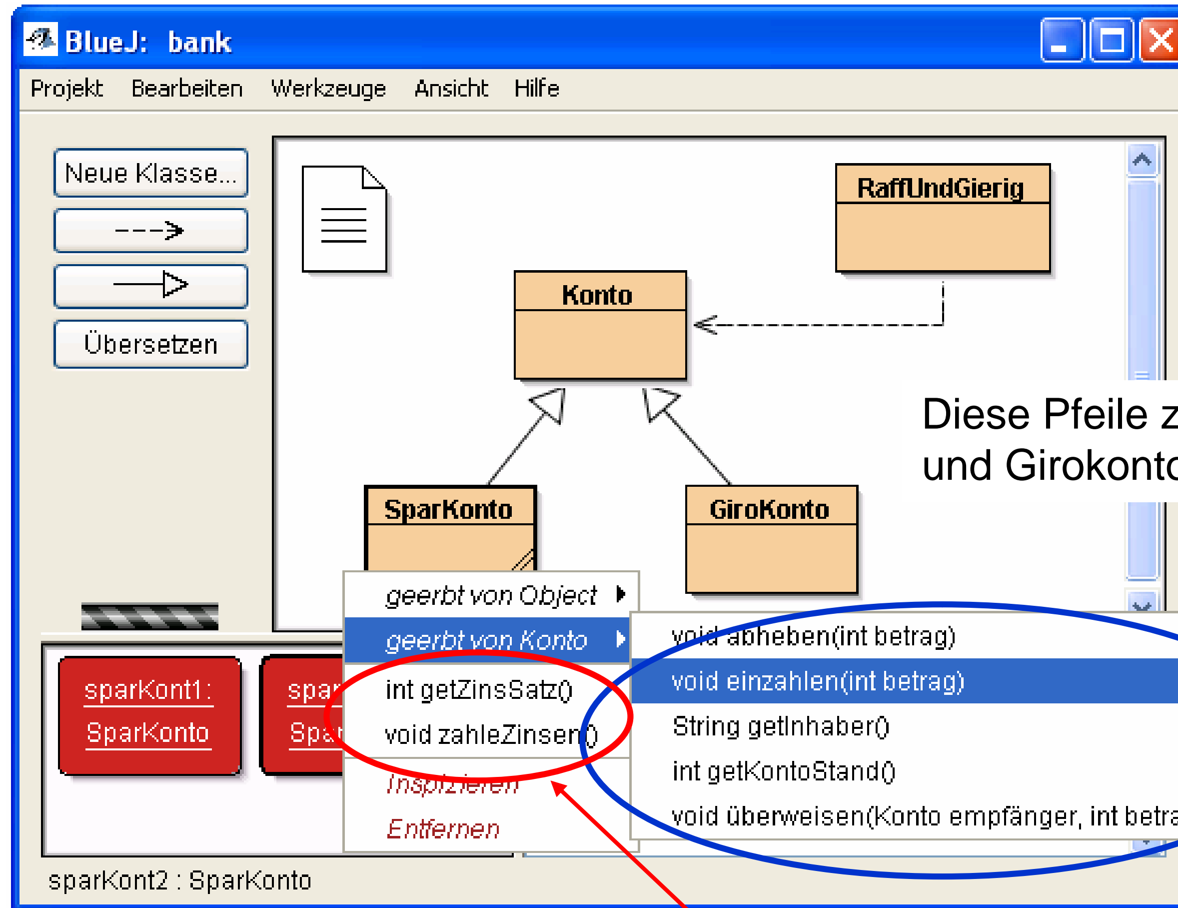
this – *ich*

- this steht für das aktuelle Objekt
- this(), this(…) für Konstruktoren der aktuellen Klasse.

```
 */
public Konto(){
  /* Der folgende Code muss in den anderen
   * Konstruktoren auch ausgeführt werden.
  */
  kontoNummer = kontenZaehler;
  kontenZaehler++;
}

public Konto(String name){
      this();   // Ruft den Konstruktor ohne Parameter auf
      inhaber = name;
}

public Konto(String name,int betrag){
      this(name);     // Rufe Konstruktor mit String-Parameter
      kontoStand = betrag;
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Die Unterklasse in BlueJ

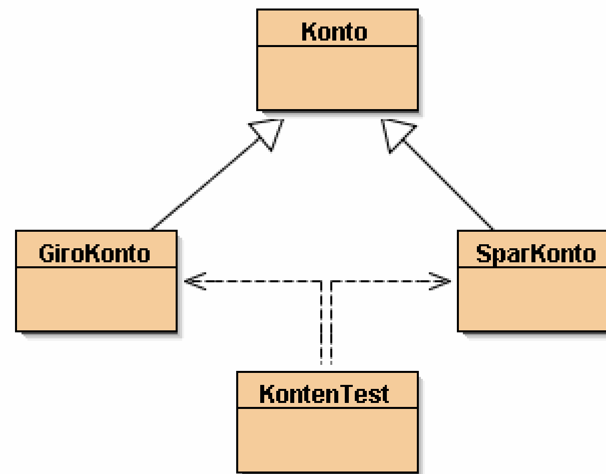


Diese Pfeile zeigen an, dass **SparKonto** und Girokonto Unterklassen von **Konto** sind.

Jedes Objekt der Teilklasse
- hat zusätzliche Methoden
- erbt alle Methoden der Oberklasse:

©H. Peter Gumm, Philipps-Universität Marburg

Eine Klasse mit Unterklassen



Klasse Konto mit Unterklassen
GiroKonto und
SparKonto
und eine Klasse, die die letzteren benutzt.

```
public class GiroKonto extends Konto{
//==========================================
// Objekt-Felder
//==========================================
   private int kreditLimit;
   private int ueberziehungsZinsSatz = 10;

//==========================================
// Konstruktor(en)
//==========================================
    public GiroKonto(){
       super();
    }
```

```
//==========================================
// Klassen-Felder und -Methoden
//==========================================
  /**
   * Der sparZinsSatz ist für alle Sparkonten
   * gleich, daher ein Klassenfeld (static)
   */
  private static int sparZinsSatz = 3;
//==========================================
// Objekt-Felder
//==========================================

   /** Keine weiteren Objektfelder */
```


©H. Peter Gumm, Philipps-Universität Marburg

Overriding: über-definieren

- Methoden der Oberklasse können in der Unterklasse verändert werden:
  - z.B. kann die Methode **abheben** in der Unterklasse **SparKonto** neu definiert werden:

```
class SparKonto extends Konto{
          . . .
  void abheben(int betrag){
     if (getKontoStand() >= betrag)
             super.abheben(betrag);
  }
        . . .
```

Nur falls (**if**)
(**kontoStand** ≥ **betrag**)
ist führe die Aktion aus.

Die Methode **abheben** der Oberklasse, hier **Konto**.

©H. Peter Gumm, Philipps-Universität Marburg

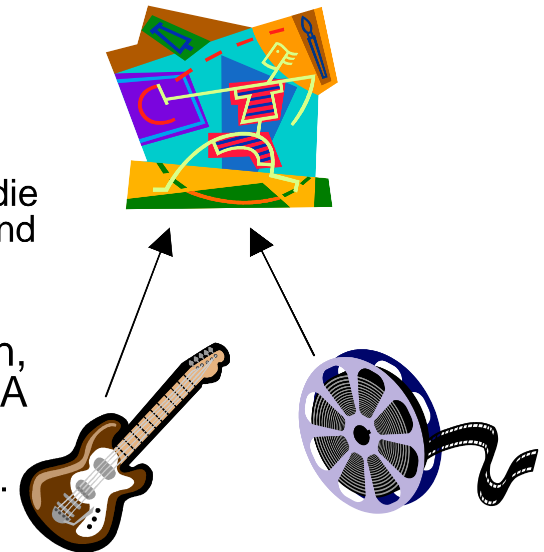
Klasse - Unterklasse

- Wann sollte man eine Klasse als Unterklasse einer bestehenden Klasse modellieren?
  - Nur wenn es begrifflich Sinn macht
    - Jeder Kreis ist eine Figur,
    - jede Figur ist ein geometrisches Objekt,
    - jedes Sparkonto ist ein Konto,
    - jeder PKW ist ein Fahrzeug
  - Nur wenn jedes Objekt der neuen Klasse als spezielles Objekt der bereits bestehenden Klasse angesehen werden kann
    - Quadrat ist ein spezielles Rechteck,
    - ein Kreis ist eine spezielle Ellipse,
    - ein Sparkonto ist ein spezielles Konto,
    - eine ganze Zahl ist eine spezielle rationale Zahl
  - Nur wenn die Methoden der Oberklasse auch in der Unterklasse Sinn machen.
    - makeVisible() der Klasse Figur macht auch Sinn für jeden Kreis,
      länge(), breite() und fläche() der Klasse Rechteck machen auch Sinn für Quadrate,
      referenzPunkt() der Klasse GeometrischesObjekt macht auch Sinn für Kreise

©H. Peter Gumm, Philipps-Universität Marburg

Fallstricke

- In vielen Fällen haben Objekte der Unterklasse zusätzliche Felder
  - Ein `SparKonto` ist ein `Konto` mit zusätzlichen Feldern, z.B. `sparZins`, `letzterZinstag`, ...
  - Ein `PKW` ist ein `Fahrzeug` mit zusätzlichen Feldern, z.B. `personenZahl`, ...
  - Eine `MusikCD` bzw. ein `Film` ist ein spezielles `MedienKunstwerk`, die neben `autor`, `titel` und `herstellungsJahr` noch `titelListe` und `musikGenre` bzw. `schauspielerListe` und `kameramann` hat.



- Allein die Tatsache, dass Objekte der Klasse B mehr Felder haben, als Objekte der Klasse A macht B noch nicht zur Unterklasse von A
  - Ein `Quadrat` hat eine `länge`, ein `Rechteck` hat `länge` und `breite`. Ein `Rechteck` ist aber kein spezielles `Quadrat` – ganz im Gegenteil !
  - Eine `RationaleZahl` hat `zähler` und `nenner`, eine `GanzeZahl` nur `zähler`. `RationaleZahl`en sind aber nicht spezielle `GanzeZahl`en – im Gegenteil !

©H. Peter Gumm, Philipps-Universität Marburg

Eingebaute Java-Klassen *erweitern*

- Klassen erweitern, heißt Unterklassen bilden
  - Unterklasse erbt alle Felder und Methoden der Oberklasse
    - `SparKonto` und `GiroKonto` erben Funktionalität von `Konto`
  - Java hat eine umfangreiche Klassen-Bibliothek
  - Fast jede)* der Klassen kann man erweitern:
    - Um ein Applet zu schreiben beerbt man die Klasse `Applet`. Sie stellt Methoden bereit,
      - ein Applet im Browser darzustellen
      - es zu starten bzw. zu stoppen, wenn das Browserfenster verändert wird
      - Auf der Leinwand des Applets zu schreiben, zeichnen, Buttons und Menüs anzubringen
    - Um User-Interfaces zu schreiben beerbt man die Klassen
      - `Graphics`, `Event`, `Menu`, `Color`, `Window`, `Button`, ...

    ... die sich in dem in dem Paket `java.awt` befinden.

)* alle Klassen, die nicht als `final` deklariert wurden

©H. Peter Gumm, Philipps-Universität Marburg

Pakete

- Java Klassen, die zusammengehören, sind in Paketen organisiert
- Pakete sind hierarchisch organisiert
  - java.awt
    - java.awt.color
    - java.awt.image
      - java.awt.image.renderable
    - java.awt.font,
    - ...
- Wichtige Pakete sind
  - java.lang
    - Enthält alle Klassen, des Sprachkerns :
      - Object, System, Math, String, ...
  - java.applet
    - Es enthält die Klasse Applet
  - java.io
    - Klassen um Dateien zu lesen und zu schreiben
  - java.util
    - Behälterklassen (collections),
    - formatierte Eingabe (Scanner), Datum, Zeit, etc.
  - java.net
    - Kommunikation, Netzwerkdienste
  - java.awt
    - Abstract Windowing Toolkit. Sammlung von Klassen für Benutzeroberflächen

©H. Peter Gumm, Philipps-Universität Marburg

Pakete öffnen

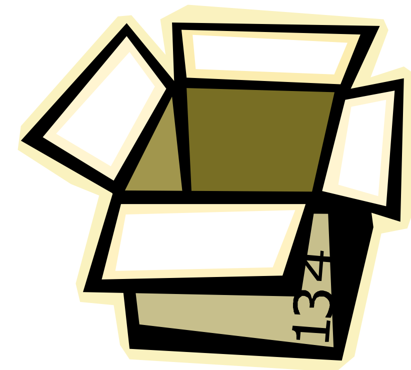
- Eine Klasse benennt man, indem man den Paketnamen davorsetzt
  - java.awt.Graphics
  - java.awt.event.MouseAdapter

- Man kann auch ein Paket „öffnen“ und darin enthaltenen Klassen verfügbar machen
  - `import java.awt.Graphics ;`

    Macht alle verfügbaren Methoden und Felder der Klasse Graphics sichtbar

- Um alle Klassen eines Pakets zu öffnen, benutzt man *wildcards*:
  - `import java.awt.image.* ;`
  - `import java.awt.* ;`
  - `import javax.swing.* ;`

- Die import-Anweisungen
  - müssen zu Beginn der Klasse stehen
  - sie importieren eigentlich nichts, sie veranlassen nur den Compiler an den entsprechenden Stellen zu suchen, wenn er auf etwas stößt, was er nicht kennt.



import

- **import** *paket.Klasse*;
  - Öffnet den Namensraum einer Klasse.
  - Alle enthaltenen als public deklarierten Klassen, Felder und Methoden werden sichtbar.
- **import paket.***;
  - öffnet den Namensraum aller Klassen im **paket**
    - Bsp.: import java.awt.event.* ;
    - import java.applet.* ;

- **import**–Anweisung muss erste Anweisung sein – noch vor "**class**"

```
import java.awt.color.*;
import java.io.*;
class MyClass{ … }
```

- **import** importiert keine Daten
  - macht lediglich Namen sichtbar und benutzbar.

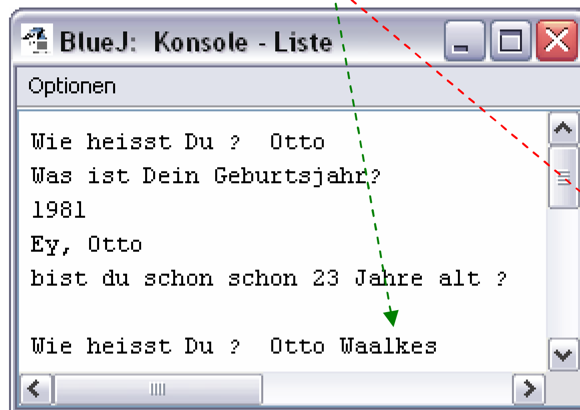
©H. Peter Gumm, Philipps-Universität Marburg

Beispiel: *Scanner* in *java.util*

- Ein *Scanner* eröffnet einen *Eingabestrom* auf eine Datei oder auf das Terminal
  - im Paket java.util
- mit next() liefert der Scanner das nächste Wort im Input
- mit nextInt() den nächsten int
- kommt etwas anderes, wird ein Fehler gemeldet

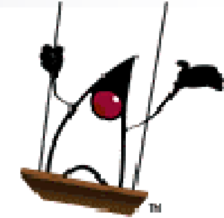
```
import java.util.Scanner;
import java.io.PrintStream;

public class Scan{

    public static void main(String[] args){

        PrintStream aus = System.out;
        Scanner eingabe = new Scanner(System.in);

        aus.print("Wie heisst Du ?  ");
        String name = eingabe.next();
        aus.println("Was ist Dein Geburtsjahr?");
        int jahr = eingabe.nextInt();
        aus.println(
            "Ey, "+ name + "\nbist du schon schon "
            +(2004-jahr)+" Jahre alt ?\n");
    }
}
```

InputMismatchException:
null (in java.util.Scanner)

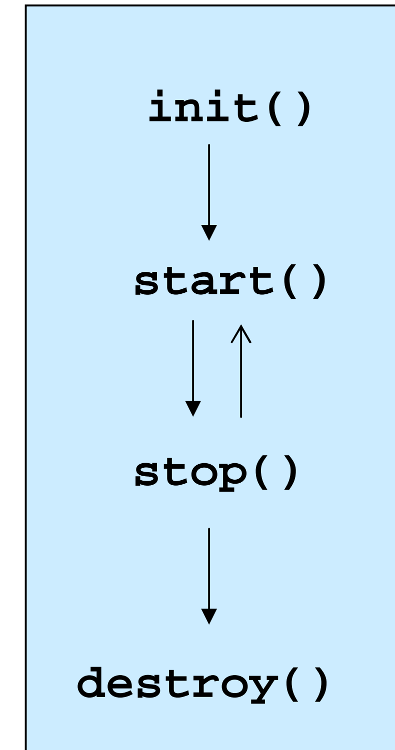


©H. Peter Gumm, Philipps-Universität Marburg

Die Klasse Applet



- WWW-Browser benutzen die Klasse Applet
  - Der Browser erzeugt zunächst ein Applet
    `new Applet( … )`
  - das Applet wird initialisiert durch die Methode
    `init()`
  - und gestartet mit
    `start()`
  - Wechselt der Browser zu einer anderen Webseite, so ruft er die Methode
    `stop()`
  - Kommt er zurück, erfolgt wieder ein
    `start()`



- Jedesmal, wenn das Applet neu zu zeichnen ist
  - beim Start
  - wenn das Browserfenster Größe oder Position ändert,

  ruft der Browser die als **`public`** erklärte Methode

  `paint()`

  mit Signatur `void paint(Graphics g)` auf.

`paint()`

- Ein Applet programmieren heißt:
  - Einige der genannten Methoden re-definieren.

©H. Peter Gumm, Philipps-Universität Marburg

Wir beerben die Klasse Applet

- Die Methoden (paint, init, start, stop, destroy) werden von außen – von dem Browser – aufgerufen.
  - Klasse und ihre Methoden müssen daher *public* sein

```
import java.applet.Applet;
import java.awt.*;

public class Äpfelchen extends Applet
{
    public void paint(Graphics g){
        g.drawLine(50,50,200,300);
        g.drawString("Von hier ...",50,50);
        g.setColor(Color.blue);
        g.drawString("... bis hier",200,300);
        g.drawOval(100,100,50,80);
    }

}// Ende der Klasse Aepfelchen
```

Klasse übersetzt - keine Syntaxfehler   *gespeichert*

- Wir beerben die Klasse
  - java.applet.Applet
- Wir redefinieren
  - paint()
- Wir benutzen die Klassen
  - java.awt.Graphics
  - java.awt.Color

©H. Peter Gumm, Philipps-Universität Marburg

Ausführen in BlueJ

- Nach dem Kompilieren hat das Kontextmenü der Klasse „Äpfelchen“ eine Auswahl: *Run Applet*
- wir wählen eine Dimension aus und
- sehen das Ergebnis per **`appletviewer`** dargestellt

Applet in Webpage einbinden

Appl - Notepad

File Edit Format View Help

```
<html>
<head><title>Ein erstes Applet</title>
</head>

<body>
<h1>Nur ein kleines Äpfelchen</h1>

<applet code=Äpfelchen width=300 height=200>
</applet>

<p>Tja, das wars ...
<body>
</html>
```

- Ein Applet lässt sich leicht in eine Webseite einbinden
- HTML-Marke: <Applet> </Applet>
- Wichtigste Parameter: code, width, height
- Code-Datei hier: `Äpfelchen.class`

©H. Peter Gumm, Philipps-Universität Marburg

Nicht gerade spektakulär ...

Die HTML-Datei - in verschiedenen Browsern