Sichtbarkeit

Schnittstellen, Protokolle, Treiber,
Information hiding, Datenkapselung,
Zugriffsattribute, Pakete, Polymorphie,
Java-Interfaces, Klassenhierarchie,

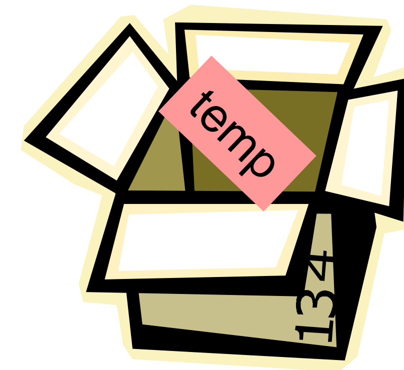
Information hiding

- Anwendungen stellen bereit
  - Informationen
  - Dienste
- Programme verwenden
  - Variablen
  - Methoden
  - Klassen
- Einige davon sind für den Anwender bestimmt
  - `main`
  - Menüfunktionen
- Andere sind nur interne Hilfskonstrukte
  - lokale Variablen
  - Hilfsmethoden
  - Hilfsklassen
- Jegliche interne Information sollte dem Anwender verborgen und unzugänglich bleiben

©H. Peter Gumm, Philipps-Universität Marburg

Versteck: Lokale Variablen

- Lokale Variable
  - Innerhalb eines Blockes deklarierte Variable
  - Außerhalb des Blockes ist sie nicht sichtbar




```
int x=4, y=17;
{ int temp;
   temp = x;
   x = 17;
   y = temp;
}
System.out.println(
  "x ="+x+" y ="+ y +" temp ="+temp);
```

  - Blöcke sind Datenschachteln

Fehler
`temp` ist nur innerhalb des Blockes sichtbar

©H. Peter Gumm, Philipps-Universität Marburg

Versteck: Methodenrumpf

- Der Rumpf einer Funktionsdefinition ist ein Block.
  - Im Rumpf definierte Variablen sind außen nicht sichtbar
- Schnittstellen von Methoden sind
  - Parameter
  - Rückgabewert.

```
int myMethod( int m , int n )
{
  int temp;
  ... tuWas...
  return ergebnis ;
}
```

Versteckt
dem Benutzer
der Methode unsichtbar

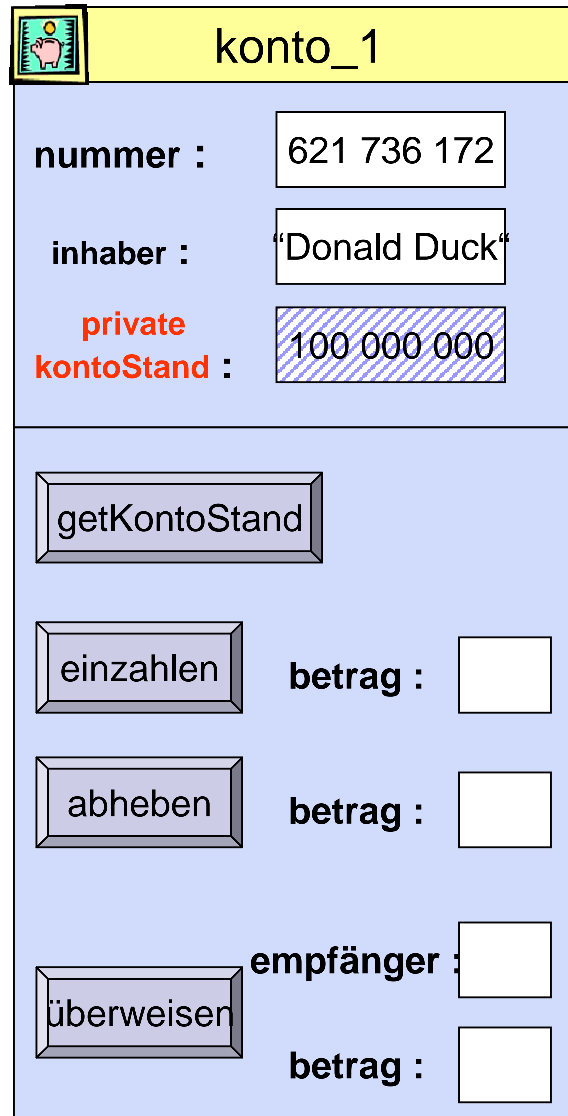
Schnittstellen

©H. Peter Gumm, Philipps-Universität Marburg

Private Felder

Kein direkter
Zugriff mehr

Der einzige Weg,
Um **kontoStand**
zu lesen bzw.
zu ändern



- Mit dem Schlüsselwort
  - private

  verbirgt man Felder einer Klasse nach aussen.
- Methoden anderer Klassen können nicht auf diese Felder zugreifen
- Objekte der gleichen Klasse haben Zugriff
- Beispiel Konto:
  - Selbst wenn wir das Feld `kontoStand` nach außen verstecken, können wir von einem anderen Konto noch zugreifen – z.B. beim Überweisen.

©H. Peter Gumm, Philipps-Universität Marburg

Weitere Zugriffsattribute

- public Felder sind explizit freigegeben
  - überall sichtbar
  - ggf. muss man die definierende Klasse benennen (importieren)
- Felder ohne Zugriffsattribute
  - Zugriffsattribut heißt auch „package"
  - in jeder Klasse des gleichen Pakets sichtbar
  - nicht in anderen Paketen
- protected Felder und Methoden
  - in der Klasse und
  - in allen Unterklassen sichtbar
  - Die genaue Spezifikation ist relativ kompliziert. Wer es genau wissen will:
    - http://java.sun.com/docs/books/jls/second_edition/html/names.doc.html#62587
- private Felder und Methoden
  - nur in der definierenden Klasse sichtbar
  - nicht in Unterklassen

niedrigster

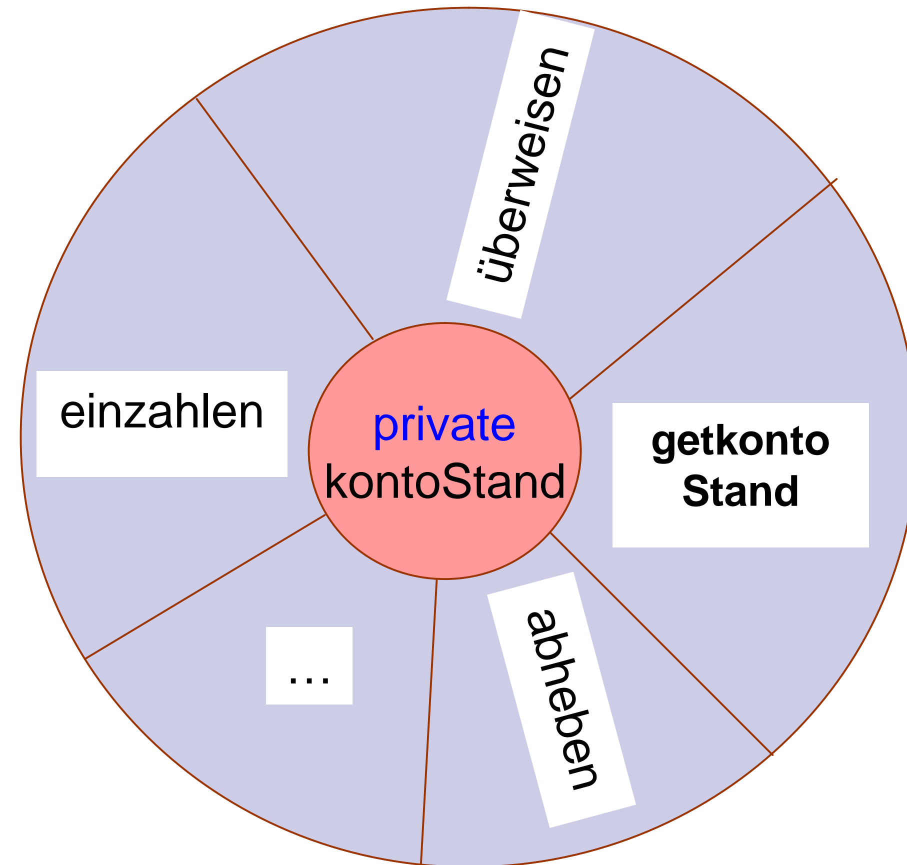
Zugriffs-Schutz

höchster

©H. Peter Gumm, Philipps-Universität Marburg

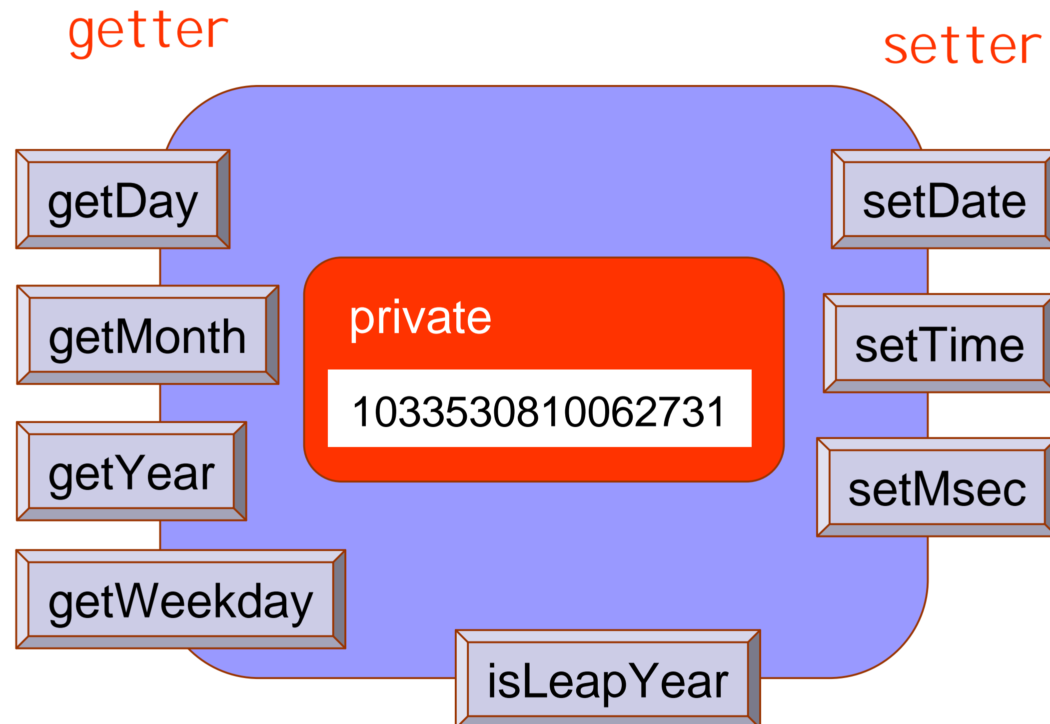
Datenkapselung

- Datenobjekte werden vor direktem Zugriff geschützt
  - private class
  - private method
  - private field
- Öffentliche Methoden definieren Benutzer-Schnittstelle
  - public class
  - public method
  - public field



Beispiel: Datum

- **UNIX repräsentiert Datum durch Millisekunden seit 1.1.1970 GMT**
- Schnittstelle: Zugriffsmethoden
  - *getter* und *setter*
    - getDay(), getMonth(), getYear()
    - getWeekday()
    - setDate(), setTime(), setMsec()
    - isLeapyear()
- Datenkapselung:
  - Verstecken der Repräsentation
  - Verstecken von Hilfsmethoden
- Vorteil
  - Repräsentation kann man ändern
    - z.B.(tag,monat,Jahr,std,min,sec,ms)
  - Zugriffsmethoden bleiben
  - Anwendungsprogramme müssen weiter funktionieren



©H. Peter Gumm, Philipps-Universität Marburg

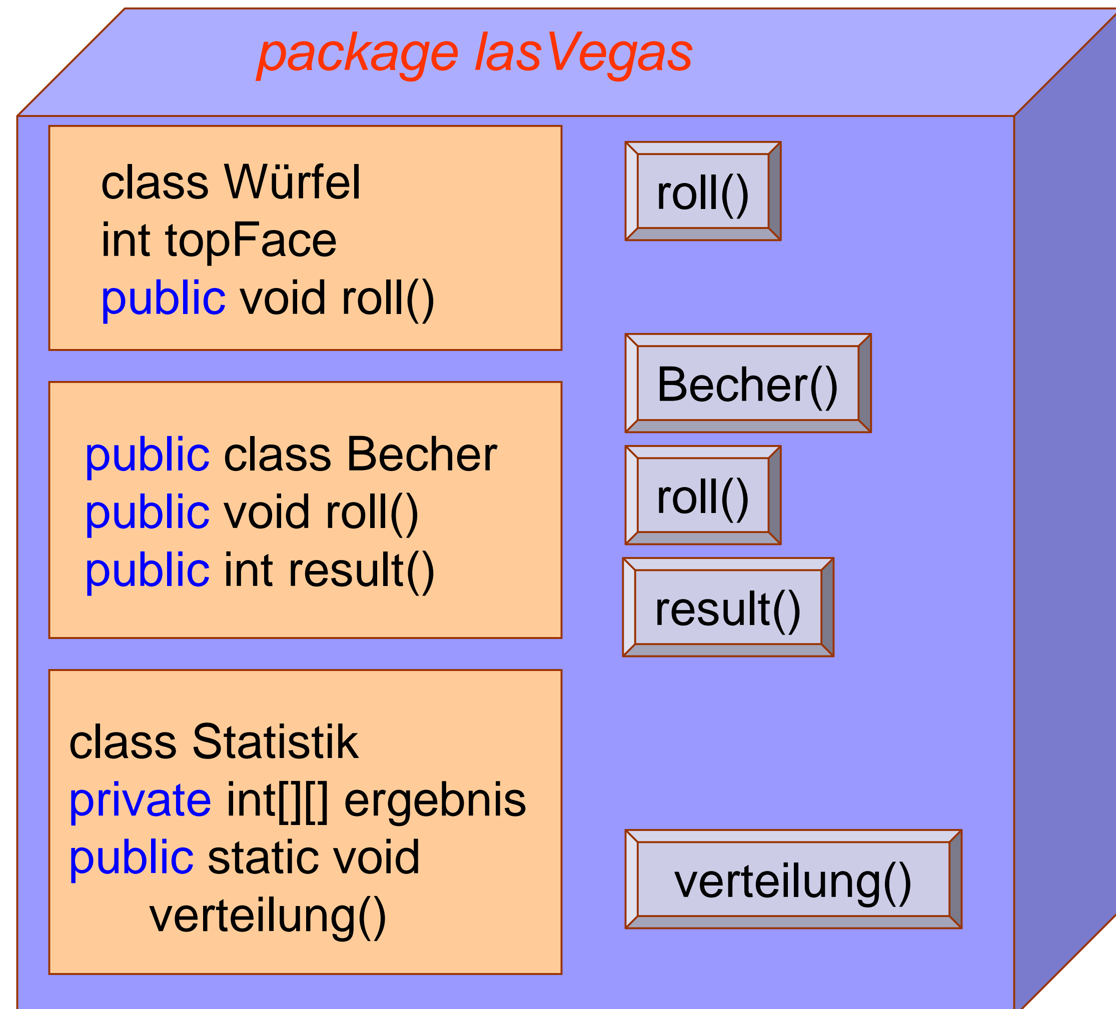
Pakete

- Java Anwendungen bestehen aus vielen Klassen
- Zusammengehörige Klassen werden zu einem **Paket** zusammengefasst
- Jede Klasse gehört zu einem Paket
  - Wenn nichts gesagt wird, zu **`default`** .
- Um eine Klasse einem Paket hinzuzufügen, muss die erste Zeile lauten:
  - package *paketName* ;
- Auf vielen Plattformen hat man die Entsprechung
  - class Meine ≅ Datei Meine.java
  - .. nach der Compilation: Meine.class
  - package meinPaket ≅ Verzeichnis meinPaket

©H. Peter Gumm, Philipps-Universität Marburg

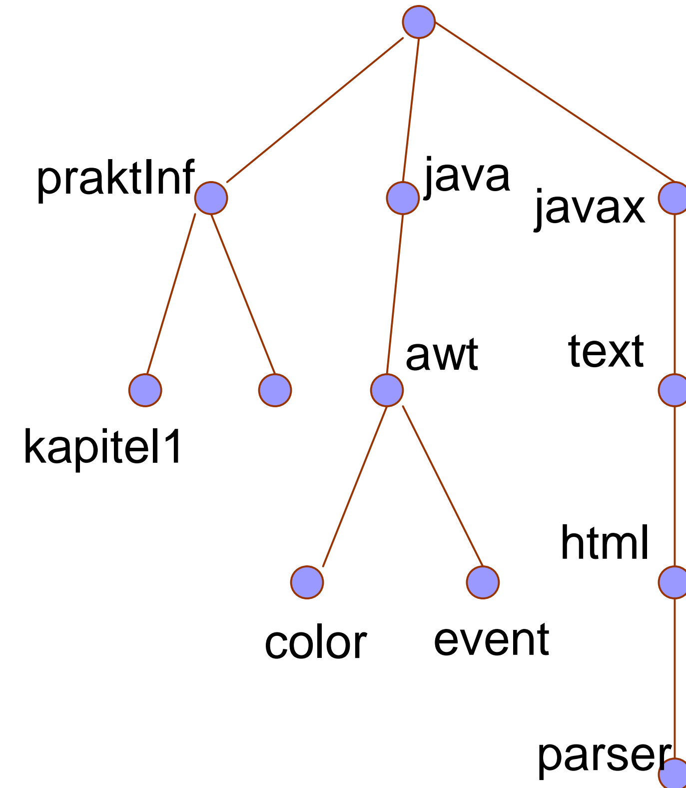
Paketzugriff

- Klassen, Methoden und Felder in fremden Paketen sind nicht zugreifbar
- Ohne spezielle Attribute nur im eigenen Paket sichtbar
  - (package private)
- Abhilfe:
  - Als public deklarieren
- Vergleiche javadoc !



©H. Peter Gumm, Philipps-Universität Marburg

Pakethierarchie



- Pakete können hierarchisch organisiert werden.
- Der volle Name besteht aus dem Pfad zu dem Paket
  - Beispiele:
    - java.awt.event, java.awt.color
    - javax.swing.text.html.parser
    - praktInf.kapitel1
- Die Wurzeln der Hierarchien werden, durch Semikolon getrennt, in der Variablen `CLASSPATH` mitgeteilt
  - ```
    CLASSPATH=.; C:\Programs\bluej\examples ;
    C:\mypackages;
    C:\DokumentsandSettings\praktInf\kapitel1
    ```


©H. Peter Gumm, Philipps-Universität Marburg
    ```

Referenzieren

- Pfad nötig, um Klassen in anderen Paketen anzusprechen
  - Compiler und JRE suchen unter den in CLASSPATH angegebenen Einstiegspunkten
- Klassen werden durch Voranstellen des Paketnamens eindeutig:
  z.B. java.util.Date :
  - java.util ist das Paket
  - Date ist die Klasse
- Wichtig:
  - Klasse Date muss public sein !

Klasse Date im Paket java.util

```
import java.util.Date;

class Wann{
    static String wasHamwernHeute(){
        Date heute = new Date();
        return heute.toString();
    }
}
```

```
String result = "Wed Sep 25 17:27:06 CEST 2002"
```

Oder direkt im Codepad:

```
(new java.util.Date()).toString()
"Tue Sep 12 12:09:01 CEST 2006"   (String)
```


©H. Peter Gumm, Philipps-Universität Marburg

Internationale Pakete

- Klassen werden oft im Internet bereitgestellt

- Mit der Internet-Adresse im package-Namen erreicht man Eindeutigkeit

  - **`de.unimarburg.informatik.ftp.javaKram.Würfel`**

  bezeichnet die

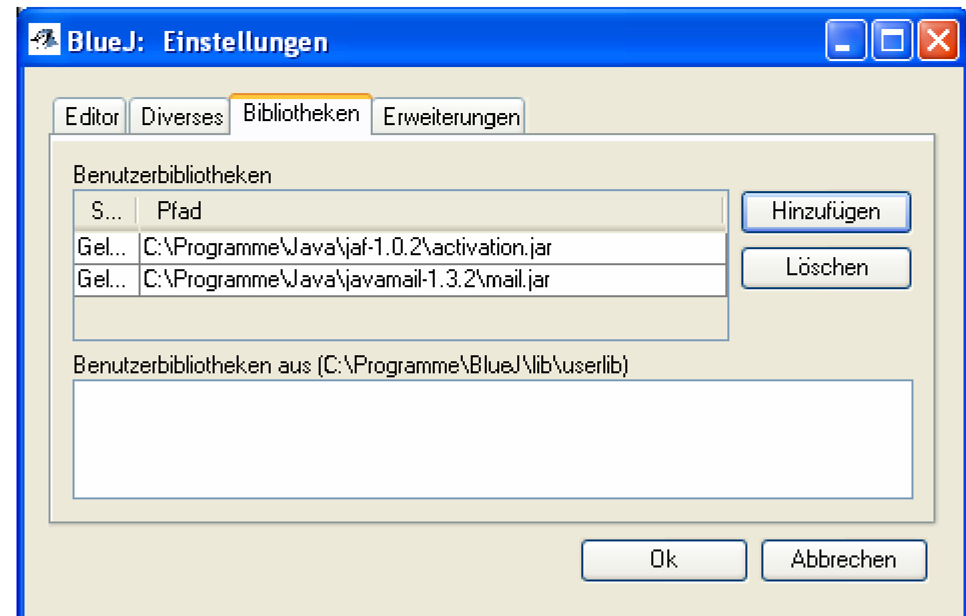
  - Klasse **`Würfel`**
  - im Verzeichnis **`javaKram`**
  - auf dem Rechner **`ftp`**
  - In der Subdomain **`uni-marburg`**
  - der Domain **`de`**

- Das Paket muss vor der Benutzung auf den lokalen Rechner geladen und installiert werden

©H. Peter Gumm, Philipps-Universität Marburg

Paket-Registrierung in BlueJ

- Beispiel:
  - Programmierung von email-Versand in Java
  - wir brauchen geeignete Klassen aus
    - javamail-API und in dem
    - java-activation-framework
  - bei java.sun.com herunterladen und entpacken.
  - Verzeichnisse enthalten jar-Dateien
    - `mail.jar`, bzw
    - `activation.jar`
  - in den Klassenpfad aufnehmen
    - entweder durch Setzen von CLASSPATH
    - oder in Bluej registrieren Einstellungen →Bibliotheken → Hinzufügen



©H. Peter Gumm, Philipps-Universität Marburg

Brief basteln und senden

BriefVersand

Klasse Bearbeiten Werkzeuge Optionen

Übersetzen Rückgängig Ausschneiden Kopieren

```
1 import java.io.*;
2 import java.net.InetAddress;
3 import java.util.*;
4
5 import javax.activation.*;
6 import javax.mail.*;
7 import javax.mail.internet.*
8
9 public class BriefVersand{
10
```

Klasse übersetzt - keine Syntaxfehler

```
public class BriefVersand{

  public static void main (String[] args)
           throws IOException, MessagingException{
    // Die Bestandteile des Briefes
    String server  = "mailhost.mathematik.uni-marburg.de";
    String from    = "bigBoss@rtl.de";      // beliebig
    String to      = "hugo@hotmail.com";    // Empfänger
    String subject = "Sie haben frei";      // Betreff
    String body    = "Sie waren nicht erfolgreich - sie haben
                   + "Das Taxi wartet.\n Machen Sies gut"
                   + "Big Boss";

    //Sitzung generieren
    Properties props = System.getProperties();
    props.put("mail.smtp.host",server);
    Session session = Session.getDefaultInstance(props,null);

    // Botschaft zu einer Mime-message zusammenpacken
    MimeMessage msg = new MimeMessage(session);
    msg.setFrom(new InternetAddress(from));
    msg.setRecipients(Message.RecipientType.TO,
                  InternetAddress.parse(to,false));
    msg.setSubject(MimeUtility.encodeText(subject,"iso-8859-1
    msg.setHeader("X-Mailer","SendEmail");
    msg.setText(body,"iso-8859-1");
    msg.setSentDate(new Date());
    // Paket absenden:
    Transport.send(msg);
  }
}
```


©H. Peter Gumm, Philipps-Universität Marburg

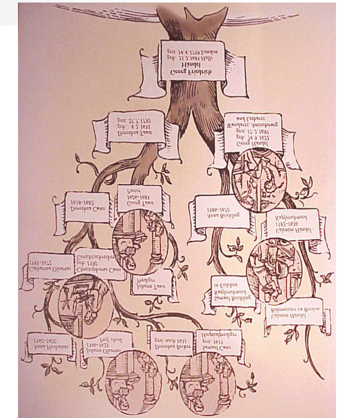
Standard Pakete

- gehören zum offiziellen Sprachumfang von Java:
  - **`java.lang`**
    - enthält u.a. die Klassen System, String, Object,
    - braucht nicht importiert zu werden

- Andere Standard Pakete
  - **`java.net`** Für Internet-Programmierung
  - **`java.text`** Zum Editieren und Formatieren von Text, Zahlen, etc.
  - **`java.awt`** (AWT=Abstract Windowing Toolkit) für GUI-Programmierung
  - **`javax.swing`** Modernere, plattformübergreifende Dialogkomponenten
  - **`java.applet`** Applets werden als Unterklasse von java.applet.Applet realisiert
  - **`java.io`** Klassen für Ein/Ausgabe
  - **`java.util`** Nützliches, wie z.B. Datum, Behälter, Listen, etc.

©H. Peter Gumm, Philipps-Universität Marburg

Die Java-Klassenhierarchie



- Ein Baum mit Wurzel `Object`
  - Jede Klasse ist Unterklasse von `Object`
- Jede Klasse wird in die Hierarchie eingehängt
  - `class MyClass extends Thread{ ... }`
  - default: unter `Object`
    - `class MyClass{ ... }`
  - ist gleich bedeutend mit
    - `class MyClass extends Object{ ... }`
- Einge Klassen dürfen nicht beerbt werden
  - Sie sind als **final** deklariert
    - `public final Boolean{ ... }`
  - Danach ist illegal:
    - `public myClass extends Boolen{ ... }`
- Jedes Paket erweitert diese Hierarchie

©H. Peter Gumm, Philipps-Universität Marburg

Klassenhierarchie

Teil der immer vorhandenen Klassen-hierarchie in

java.lang

Legende: class | final class | abstract class | final, non-instantiable

©H. Peter Gumm, Philipps-Universität Marburg

Vererbung von Methoden

- Jede Klasse ist Unterklasse von Object
- Jede Operation und jede Methode von Object ist in jeder Klasse verfügbar, z.B.:
  - `==` **(Testet Objekte auf gleiche Referenz)**
  - `equals()`
  - `toString()`
- Aber default oft nicht gut genug:
  - **equals()** testet von Hause aus auch auf gleiche Referenz
  - **toString()** gibt Klassennamen mit Referenz ausl:

```
Konto test = new Konto(„Otto");
System.out.println(test.toString());
```

  ergibt

```
Konto@a1d1f4
```

BlueJ: Terminal Window

Options

```
Kontonr.: 0
Inhaber : K. R. Ösus
Stand   : 100000000
```

- In eigenen Klassen sollten diese Methoden redefiniert werden, z.B.:

```
    public String toString(){
        return ("Kontonr.: "+nummer
                +"\nInhaber : "+inhaber);
    }

    void druckeAuszug(){
        System.out.println("\n"+toString());
        System.out.println("Stand    : "+kontoStand);
    }
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Polymorphie

- Fähigkeit verschiedene Formen anzunehmen
- Im OO-Programmieren:
  - Methoden der Oberklasse sind auch auf Objekte der Unterklasse anwendbar
  - Sie können für Objekte der Unterklasse re-definiert werden
  - Sie sollten semantisch ähnliches Verhalten aufweisen
    - `toString()` aus Objekt funktioniert für alle Unterklassen;
      dort kann man es aber aussagekräftiger implementieren
    - `druckeAuszug()` aus `Konto` funktioniert auch für Sparkonten
  - Methoden gleichen Namens können verschiedene Signaturen haben
    - `Konto()`
    - `Konto(String inhaber);`
    - `Konto(int nummer, String inhaber)`
    - `Integer parseInt(String zahl)`
    - `Integer parseInt(int zahl)`

©H. Peter Gumm, Philipps-Universität Marburg

Weitere Teile der Hierarchie

- Jedes Paket ergänzt die Klassen-hierarchie
  - Z.B. java.util

java.lang

Object

java.util

BitSet

Date

Dictionary — HashTable — Properties

Observable

Random

StringTokenizer

Vector — Stack

©H. Peter Gumm, Philipps-Universität Marburg

Schnittstellen

- **Vereinbarung über Außenverbindung von Systemen**
- **Begriffsursprung:**
  - Man will komplexe Systeme zerlegen
  - Einzelteile von verschiedenen Firmen hergestellt
  - An den *Schnittstellen* müssen Teile zueinander passen
  - Folglich muss die Schnittstelle genau spezifiziert sein
- **Notwendig, um Systeme verschiedener Hersteller zu verbinden**
  - **Informatik**
    - Computer und Monitor
    - Diskette und Laufwerk
    - Modem und Telefonsystem
  - **Verkehrswesen**
    - Räder und Reifen
    - Motor und Treibstoff (Tankstelle)
  - **Haus**
    - Mülltonne und Müllwagen
    - VHS-Kassette, Recorder

- Beispiel: Steckdose
  - Schnittstelle regelt
    - Form
    - Beschaltung (Masse, Phase,)
    - Spannung (230 ±10V , 50 Hz)
  - Hersteller von Elektrogeräten können davon ausgehen:
    - der Stecker passt
    - die Beschaltung ist wie erwartet
    - die Stromstärke ist 230 V
    - sie schwankt nur in bestimmten Grenzen

©H. Peter Gumm, Philipps-Universität Marburg

Protokolle – Schnittstellen der Kommunikation

- **Ein Protokoll regelt**
  - Sprache/Signale
  - Reihenfolge
  - Bedingungen

  einer Kommunikation

- **Protokolle verbinden**
  - Technische Geräte
    - Telefonprotokoll
  - Rechner
    - PPP, TCP/IP
  - Programme
    - ODBC, Email
  - Menschen
    - „Hallo“, „...darf ich vorstellen“, „Tschüss“
    - Anklopfen, „Herein“, „Guten Tag“
  - Staaten
    - Diplomatisches Protokoll
    - Beistandsverträge

- Technische Schnittstellen und Protokolle gehören oft zusammen
  - Telefon
  - Fax
  - RS232
  - Ethernet
  - ...
- Hardware und Treiber
  - ISDN Karte

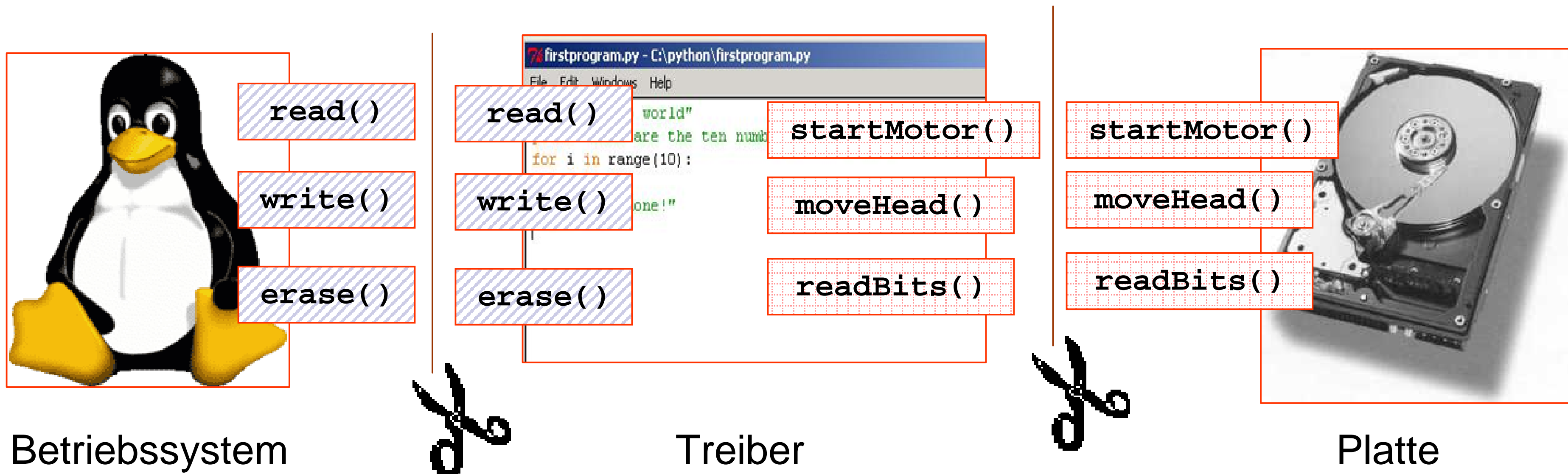
©H. Peter Gumm, Philipps-Universität Marburg

Treiber

- Treiber sind Programme um unterschiedliche Schnittstellen zu verbinden

- Oft wird eine Hardwareschnittstelle
  - Laufwerk
  - Kamera
  - Modem

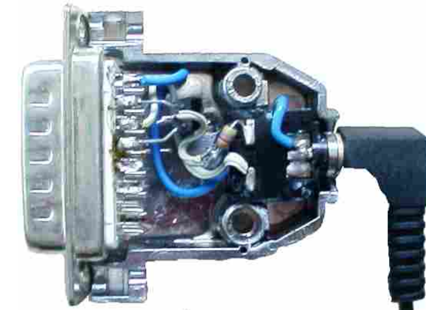
  mit einem Betriebssystem verbunden
  - Linux
  - Windows
  - MacOS



©H. Peter Gumm, Philipps-Universität Marburg

Warum Schnittstellen ?

- Eine Schnittstelle ist wie ein Vertrag
  - Wenn Deine Steckdose den richtigen Strom liefert, garantiere ich, dass mein Toaster funktioniert
- Der Vertragsnehmer sollte sich nur auf die Dinge verlassen, die versprochen sind
  - Wer Interna benutzt, die nicht im Vertrag stehen, kann hereinfallen
    - Die Spannungstoleranz war doch bisher immer nur ± 0.5 V
    - Aber heute ist mein Toaster explodiert
    - Tja, versprochen waren nur ± 10 V
  - Beim nächsten Update des Betriebssystem
    - Ein Programm, das eine undokumentierte Schnittstelle benutzt stürzt ab
    - Die Internet Kamera will nicht mehr
      - Ein neuer Treiber ist fällig
- Am besten verhindert man den Zugriff auf Interna
  - Ein Bankkunde sollte nur über die offizielle Schnittstelle auf sein Konto zugreifen können



©H. Peter Gumm, Philipps-Universität Marburg

Java-Interfaces

- Der Begriff *Interface* -(dt.: *Schnittstelle*) ist zentral für die Informatik
- Java verwendet den Begriff in einem eingeschränkten technischen Sinne
- Ein Java-Interface definiert eine Funktionalität, die eine Klasse haben soll
  - Methoden und Konstanten
  - ein Interface ist ähnlich einer abstrakten Klasse
- Eine Klasse implementiert das Interface, wenn sie die gewünschte Funktionalität bereitstellt
  - die genannten Felder und Methoden
- Eine Klasse kann mehrere Interfaces implementieren
  - Da liegt der wichtigste Unterschied zwischen Interfaces und abstrakten Klassen

Wozu interfaces

- Interfaces sehen ähnlich aus wie abstrakte Klassen

```
/**  Interface Order:
     Intendiert ist, dass jede
     Implementierung die Relation
     "less" transitiv und irreflexiv
     definiert.
*/
public interface Order
{
   /** Vergleich - transitiv und
       irreflexiv, bitte */
   boolean less(Order y);

   /** kleiner-oder-gleich */
   boolean lessEqual(Order y);
}
```

```
/**
 * Eine Figur ist ein
 * geometrisches Objekt
 * mit Fläche und Umfang.
*/
public interface Figur
{
     float fläche();
     float umfang();
}
```


©H. Peter Gumm, Philipps-Universität Marburg

Wozu interfaces

- Ein Sortieralgorithmus kann alle Daten sortieren, auf denen eine Ordnung definiert ist

```
public class SortierAlgorithmus{

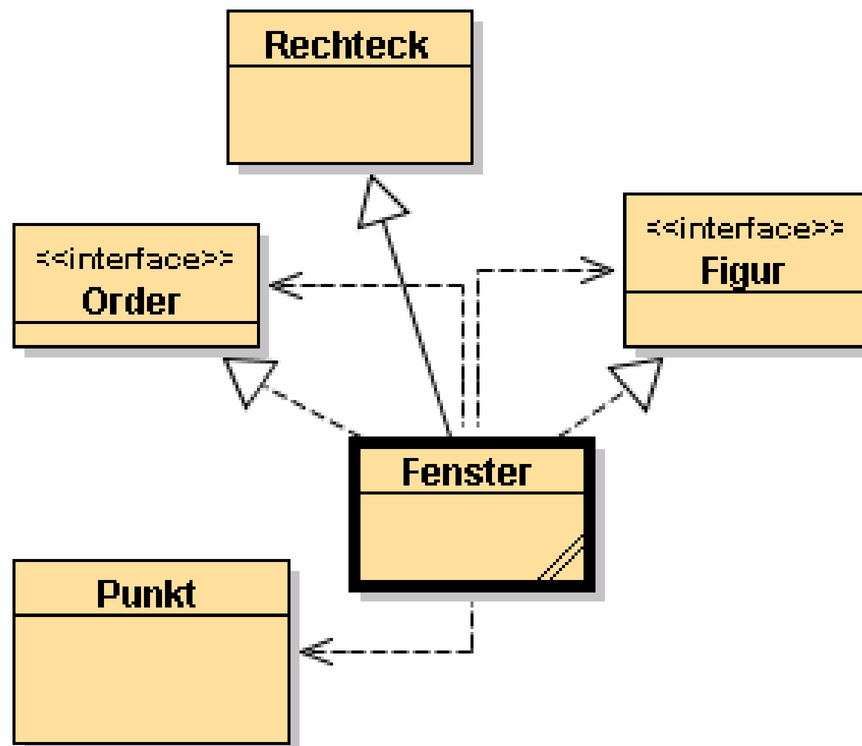
    static void bubbleSort(Order[] s){
        for(int k = s.length-1; k>0; k--)
            for (int l=0; l <k; l++){
                if(s[l+1].less(s[l]))
                    { Order temp = s[l];
                      s[l]=s[l+1];
                      s[l+1]=temp;
                    }
            }
    }
}
```

beliebige geordnete Menge

Methode des interface Order

©H. Peter Gumm, Philipps-Universität Marburg

Mehrfachvererbung



- Eine Klasse kann nur eine Superklasse haben

- In anderen OO-Sprachen ist das nicht notwendig so
  - Erbt eine Klasse von mehreren Superklassen, so spricht man von *Mehrfachvererbung*
  - multiple inheritance

- Eine Klasse kann mehrere Interfaces implemetieren.

- Eine Art Mehrfach-vererbung durch die Hintertür

©H. Peter Gumm, Philipps-Universität Marburg

Implementieren mehrerer Interfaces

```
/** Ein Fenster ist ein Rechteck in der Ebene
   Fenster sind Figuren (mit Fläche und Umfang)
   Sie sind geordnet F1 < F2 falls F2 überdeckt F1
*/
public class Fenster extends Rechteck
                     implements Figur, Order{
    private Punkt ursprung;

    Punkt getUrsprung(){ return ursprung; }

    public boolean less(Order f){
       int x1 = ursprung.getX();
       int y1 = ursprung.getY();
       int x2 = ((Fenster)f).getUrsprung().getX();
       int y2 = ((Fenster)f).getUrsprung().getY();
       return      x1 > x2
          && y1 > y2
          && x1+getLänge()   < x2+((Rechteck)f).getLänge()
          && y1+getBreite()  < y2+((Rechteck)f).getBreite();
    }
```

Rechteck wird geerbt

Zwei Interfaces werden gleichzeitig implementiert

länge und **breite** für die Implemetierung von Figur werden aus **Rechteck** geerbt

less muss als zweites Argument ein Objekt vom Typ Order haben

getUrsprung ist aber nur für **Fenster** definiert, daher ist hier ein cast notwendig. Das ist unschön, geht in Java aber nicht anders !!!

©H. Peter Gumm, Philipps-Universität Marburg

Klassendiagramme mit interfaces

- Interfaces definieren zusätzliche Hierarchien im Klassendiagramm

java.awt.image

ImageObserver

java.lang

Object

java.awt
(teilweise)

Component

Button

Canvas

Container

Panel

Window

Dialog

Frame

Scrollbar

TextComponent

TextArea

TextField

MenuComponent

MenuBar

MenuItem

Menu

CheckboxMenuItem

MenuContainer

Legende: class | abstract class | interface | implements

©H. Peter Gumm, Philipps-Universität Marburg