Informatik IIb

Objektorientierte Programmierung mit Java

Oliver Jack

Fachhochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Wintersemester 2010/11

Vorlesung 2. Klassen in Java

Lernziele dieser Vorlesung

Klassen

- Eigenschaften
- Instanzerzeugung
- Zugriff

Pakete

- Standardpakete
- Konstruktion von eigenen Paketen

Zusammenfassung

Lernziele

- Kenntnis von Klassen in Java
- Verständnis der Klassenerzeugung und des Zugriffs auf Objekte
- Kenntnis von Applets
- Kenntnis von Paketen
- Kenntnis wichtiger Standardpakete

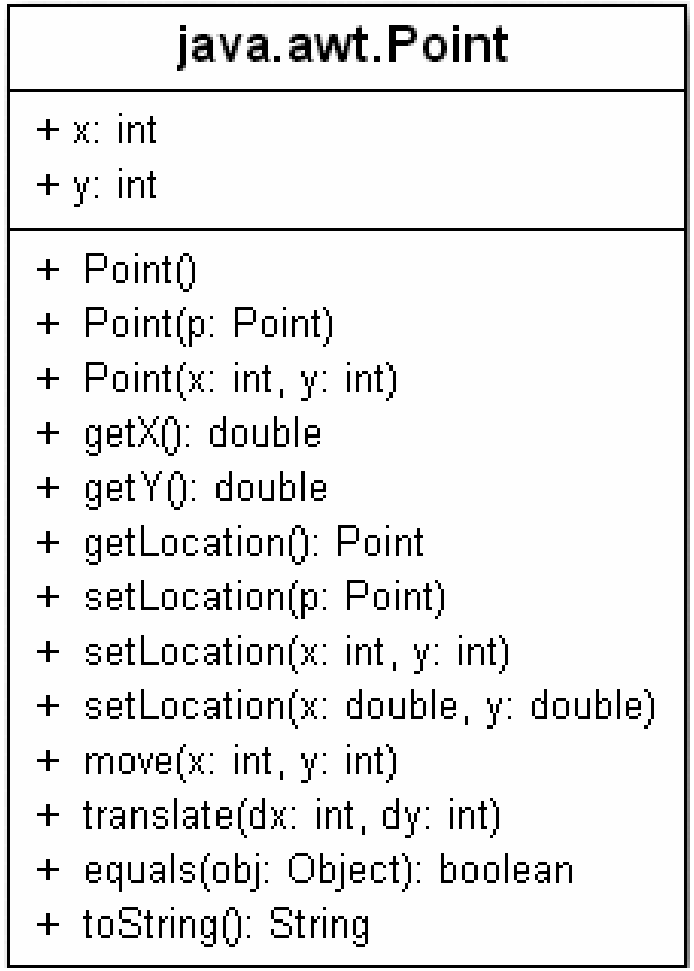
Eigenschaften einer Klasse

- Eine Klasse definiert einen Typ und beschreibt Eigenschaften der Objekte.
- Jedes Objekt ist eine Instanz einer Klasse
- Eine Klasse deklariert (im wesentlichen)
  - Attribute (was ein Objekt hat)
  - Operationen (was das Objekt kann)
- Attribute und Operationen heißen auch Eigenschaften eines Objekts.
- Operationen einer Klasse setzt die Programmiersprache Java durch Methoden (auch Funktionen genannt) um.
- Attribute eines Objekts definieren die Zustände, und sie werden durch Variablen implementiert.

Klassen aus der Standardbibliothek

Klasse Point

- Beschreibt durch die Koordinaten x und y einen Punkt in einer zweidimensionalen Ebene
- Bietet einige Operationen an, mit denen sich Punkt-Objekte verändern lassen.



Variablen

Erzeugung zur Laufzeit

Von der Klasse Point werden zur Laufzeit Exemplare erzeugt, die Point-Objekte.

Deklarieren von Variablen

- Eine Variable speichert entweder einen einfachen Wert oder einen Verweis auf ein Objekt.
- Beispiel: Deklaration einer Variablen für einen ganzzahligen Wert.

```
int i;
```

Variablen

Verweis auf ein Objekt speichert eine Referenz-Variable.

Beispiel: Variable p vom Typ Point

```
Point p;
```

- Teilt dem Compiler mit, dass diese Variable Referenzen vom Typ Point speichern soll.
- Falls es sich bei p um eine Objekt- oder Klassenvariable handelt, wird p anfangs mit der Null-Referenz (null) initialisiert, die auf kein Objekt verweist.
- Als lokale Variable hätte p keinen vorbelegten Wert, sie ist undefiniert.
- Referenztypen können nicht in primitive Typen konvertiert werden und umgekehrt.

Erzeugen einer Instanz einer Klasse

- Durch die Deklaration einer Variable mit dem Namen einer Klasse als Typ wird noch kein Exemplar erzeugt.
- Erzeugung durch Konstruktor-Aufruf mit dem Operator new.

Beispiel: Anlegen eines Objekts und Speichern der Referenz in der Variablen p

```
p = new Point();
```

Erzeugen einer Instanz einer Klasse

- Das tatsächliche Punkt-Objekt wird erst dynamisch, also zur Laufzeit, mit new erzeugt.
- Das System stellt Speicher für ein Point-Objekt bereit und speichert eine Referenz auf diesen reservierten Speicherblock in der Variablen p
- Deklaration der Variablen p und separate Erzeugung eines Exemplars der Klasse Point lassen sich, wie bei der Deklaration primitiver Datentypen, auch kombinieren.

Beispiel: Deklaration mit Initialisierung

```
double pi = 3.1415926535;
Point  p  = new Point();
```

Zugriff auf Variablen und Methoden

- Die in einer Klasse deklarierten Variablen heißen Objektvariablen, beziehungsweise Exemplar-, Instanz- oder Ausprägungsvariablen.
- Objektvariablen bilden den Zustand des Objekts.
- Der Punkt-Operator ermöglicht Zugriff auf die Methoden und Variablen eines Objekts.
- Der Punkt ist ein Operator und steht zwischen einem Ausdruck, der eine Referenz zurückgibt, und der Objekteigenschaft.

Zugriff auf Variablen und Methoden

Beispiel: Folgende Zeilen erzeugen ein Point-Objekt, speichern eine Referenz auf dieses Objekt in der Variablen p und weisen den Objektvariablen x und y Werte zu.

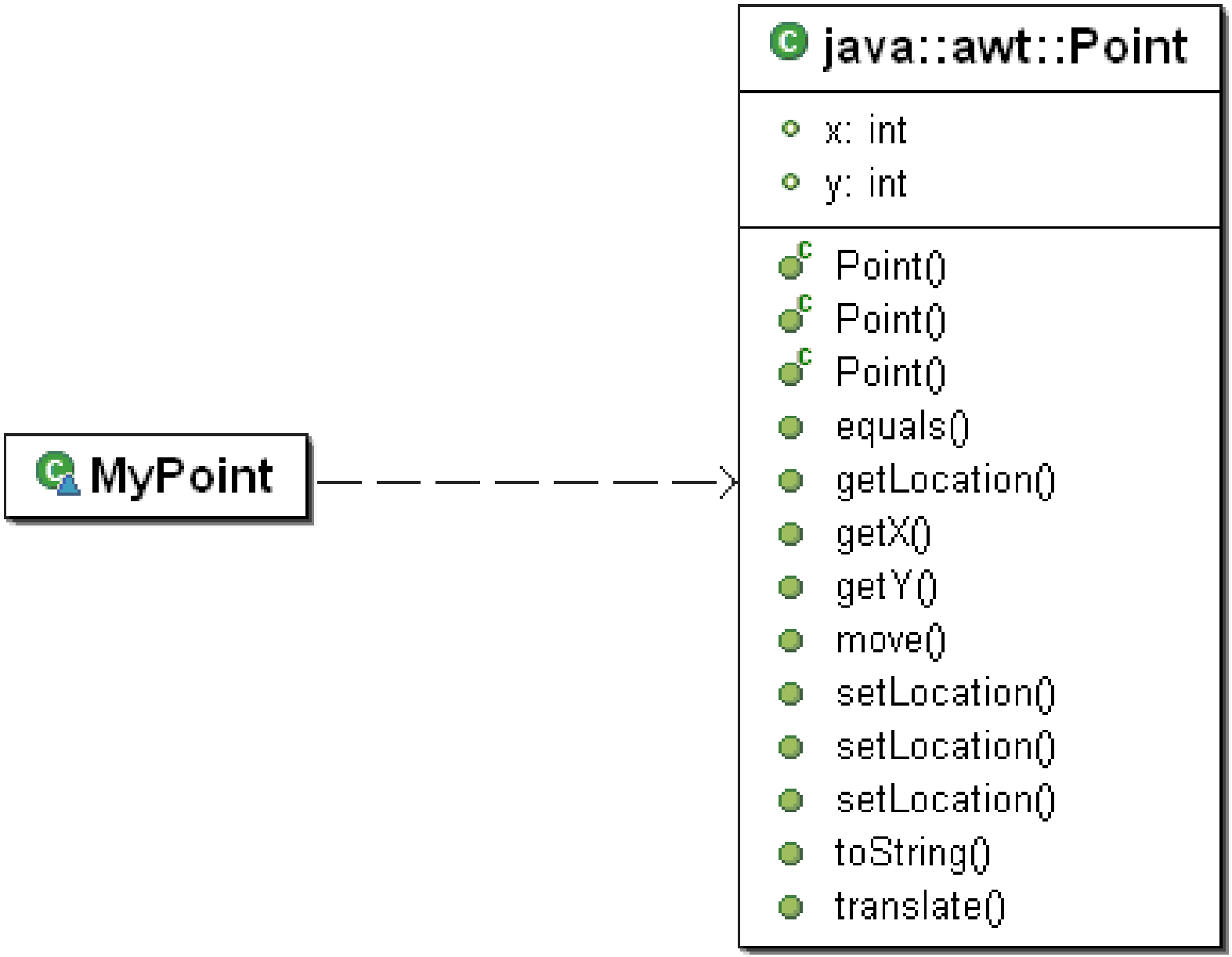
```
Point p = new Point();
p.x = 12;
p.y = 45 + p.x;
```

Beispiel: Benutzung der Point-Klasse

```
import java.awt.Point;

class MyPoint
{
  public static void main( String[] args )
  {
    Point p = new Point();
    p.x = p.y = 12;
    p.setLocation( -3, 2 );
    System.out.println( p.toString() );
   // java.awt.Point[x=-3,y=2]
//    alternativ
//    System.out.println( p );
  }
}
```

Beispiel: Benutzung der Point-Klasse



Weitergehender Zugriff auf Objekteigenschaften

- Die Funktion `toString()` liefert als Ergebnis ein String-Objekt, das den Zustand des Punkts angibt.
- Das String-Objekt besitzt selbst wieder Methoden. Eine davon ist `length()`, die die Länge der Zeichenkette liefert.

```
Point  p = new Point();
String s = p.toString();
System.out.println( s );
// java.awt.Point[x=0,y=0]
System.out.println( s.length() );    // 23
// zusammengefasst:
System.out.println( p.toString().length() );
```

Objekterzeugung ohne Variablenzuweisung

- Bei der Nutzung von Eigenschaften muss der Typ links vom Punkt immer eine Referenz, oder ein Klassenanme bei statischen Eigeschaften, stehen.
- Es funktioniert auch Folgendes

```
new Point().x = 1;
```

- Unsinnig, da zwar das Objekt erzeugt und ein Attribut gesetzt wird, anschließend das Objekt aber vom Garbage-Collector freigegeben wird.
- Sinnvoll ist jedoch

```
long size =
  new java.io.File( "datei.txt" ).length();
```

- Die Variable size enthält nun die Größe der Datei datei.txt.

Pakete

- Jede Klasse in Java ist Bestandteil eines Pakets.
- Der vollständige Name einer Klasse besteht aus dem Namen des Pakets, gefolgt von einem Punkt, dem sich der eigentliche Name der Klasse anschließt.
- Der Name des Pakets selbst kann ebenfalls einen oder mehrere Punkte enthalten.

Benutzung von Klassen eines Pakets

Damit eine Klasse verwendet werden kann, muss angegeben werden, in welchem Paket sie enthalten ist.

Zwei unterschiedliche Möglichkeiten

- Die Klasse wird über ihren vollen (qualifizierten) Namen angesprochen:

```
java.util.Date d = new java.util.Date();
```

- Am Anfang des Programms werden die gewünschten Klassen mit Hilfe einer import-Anweisung eingebunden:

```
import java.util.Date;
...
Date d = new Date();
```

Benutzung von Klassen eines Pakets

Die import-Anweisung gibt es in zwei unterschiedlichen Ausprägungen.

1. Mit der Syntax import Paket.Klasse; wird genau eine Klasse importiert. Alle anderen Klassen des Pakets bleiben unsichtbar:

```
import java.util.Date;
...
Date d = new Date();
java.util.Vector v = new java.util.Vector();
```

Benutzung von Klassen eines Pakets

Die import-Anweisung gibt es in zwei unterschiedlichen Ausprägungen.

2. Unter Verwendung der Syntax import Paket.*; können alle Klassen des angegebenen Pakets auf einmal importiert werden:

```
import java.util.*;
...
Date d = new Date();
Vector v = new Vector();
```

Benutzung von Klassen eines Pakets

- Import mit der *-Notation ist nicht ineffizienter als mit der voll qualifizierenden Notation.
- Java realisiert den Import als type import on demand, d. h. die Klasse wird erst dann in den angegebenen Paketen gesucht, wenn das Programm sie wirklich benötigt.
- Jedoch kann die voll qualifizierende Import-Notation erforderlich sein, falls mehrere mit * importierte Pakete Klassen mit dem selben Namen enthalten.
- Das Standard-Paket java.lang mit elementarer Sprachunterstützung wird bei jedem Compilerlauf automatisch importiert.

Vordefinierte Standard-Pakete des JDK (Auswahl)

Paket	*Bedeutung*
java.applet	Applets
java.awt	Abstract Windowing Toolkit (graf. Prog.)
java.io	Bildschirm- und Datei-I/O
java.lang	Elementare Sprachunterstützung
java.math	Fließkomma-Arithmetik
java.net	Netzwerkunterstützung
java.nio	Ab JDK 1.4: New I/O Package
java.security	Security-Dienste
java.sql	Datenbankzugriff (JDBC)
java.text	Internationalisierung
java.util	Diverse Utilities

Wichtige Standarderweiterungen des JDK

Paket	*Bedeutung*
javax.crypto	Kryptographische Erweiterungen
javax.imageio	Lesen und Schreiben von Bilddateien
javax.naming	Zugriff auf Namens-Services
javax.print	Unterstützung zum Drucken
javax.security.auth	Authentifizierung und Autorisierung
javax.sound	Das Sound-API
javax.swing	Das SWING-Toolkit (graf. Ben.-Oberfl.)
javax.xml	Zugriff auf XML-Dateien

Bedeutung der Paketnamen

- Jeder Teil eines mehrstufigen Paketnamens bezeichnet ein Unterverzeichnis auf dem Weg zu der gewünschten Klassendatei.
- Soll beispielsweise eine Klasse aus dem Paket `java.awt.image` eingebunden werden, sucht es der Java-Compiler im Unterverzeichnis `java\awt\image`.
- Konvention für Klassen verschiedener Hersteller: Benennung und Strukturierung der Pakete und Klassen in umgekehrten Domain-Namen.
- Beispiel: Klassen der Firma SUN, deren Domain-Name `sun.com` ist, sind in einem Paket `com.sun` oder in darunter befindlichen Subpaketen enthalten.
- Diese Konvention soll Namenskollision vermeiden.

Erstellen eigener Pakete

- Zuordnung einer Klasse zu einem Paket erfolgt durch Angabe einer package-Anweisung am Beginn des Quelltexts.
- Syntax: package Name
- Die package-Anweisung muss als erste Anweisung in einer Quelldatei stehen, so dass der Compiler sie noch vor den import-Anweisungen findet.
- Der Aufbau und die Bedeutung der Paketnamen in der package-Anweisung entspricht exakt dem der import-Anweisung.
- Der Compiler löst ebenso wie beim import den angegebenen hierarchischen Namen in eine Kette von Unterverzeichnissen auf.
- Neben der Quelldatei wird auch die Klassendatei in diesem Unterverzeichnis erstellt.

Erstellen eigener Pakete

Beispiel

Zunächst wird im Unterverzeichnis `demo` die Datei `A.java` angelegt

```
package demo;

public class A
{
  public void hello()
  {
    System.out.println("Hier␣ist␣A");
  }
}
```

Erstellen eigener Pakete

Beispiel

Im Unterverzeichnis `demo` wird weiterhin die Datei `B.java` angelegt

```
package demo;

public class B
{
  public void hello()
  {
    System.out.println("Hier␣ist␣B");
  }
}
```

Erstellen eigener Pakete

Beispiel

Im Unterverzeichnis `demo\tools` wird die Datei `C.java` angelegt

```
package demo.tools;

public class C
{
  public void hello()
  {
    System.out.println("Hier␣ist␣C");
  }
}
```

Erstellen eigener Pakete

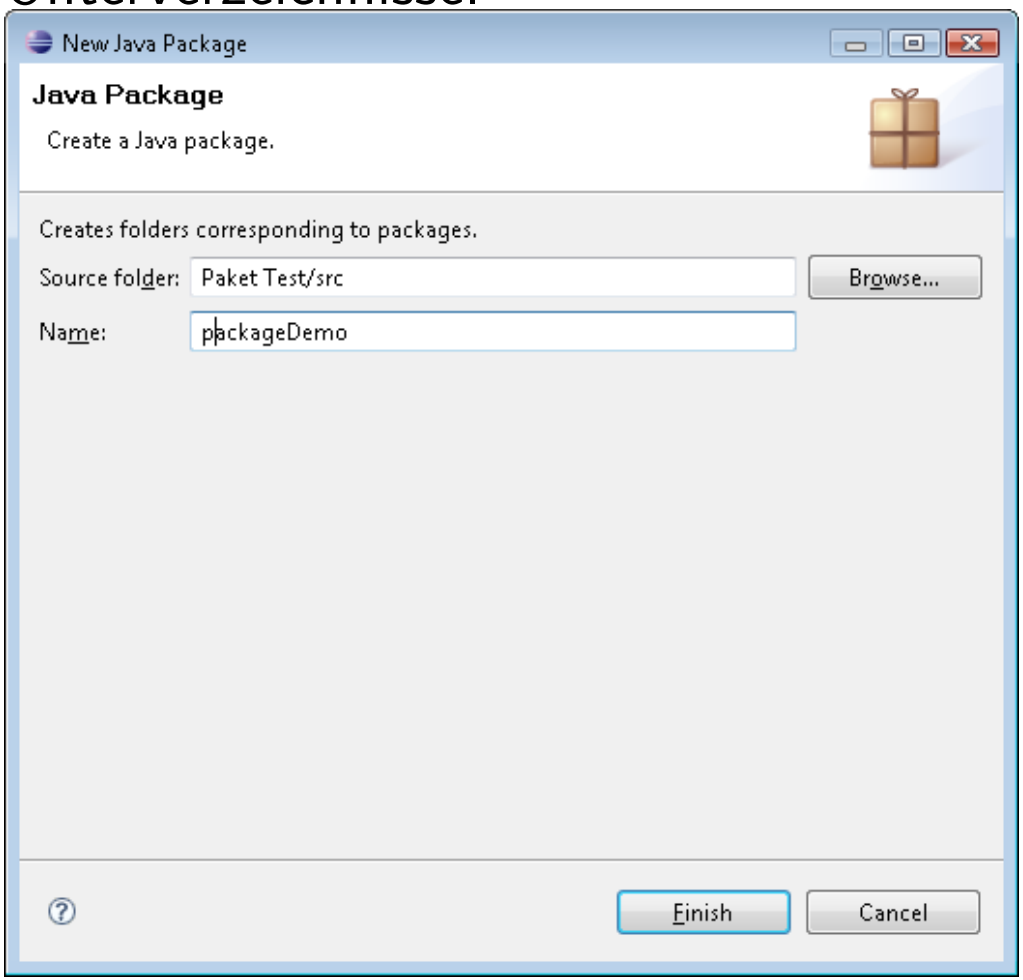
Beispiel

Nun wird im Stammverzeichnis die Datei `PackageDemo.java` angelegt

```
import demo.*;
import demo.tools.*;
public class PackageDemo
{
  public static void main(String[] args)
  {
    (new A()).hello();
    (new B()).hello();
    (new C()).hello();
  }
}
```

Erstellen eigener Pakete mit Eclipse

Eclipse erzeugt beim Anlegen von Paketen automatisch die entsprechenden Unterverzeichnisse.

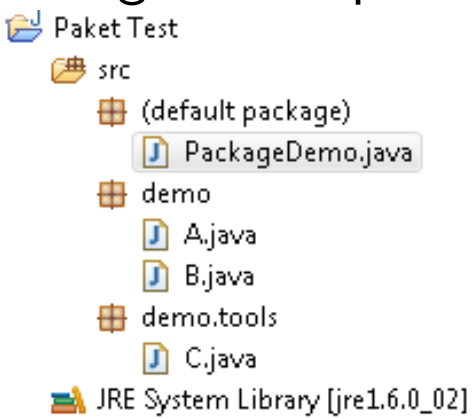


Das Default-Paket

- Zu welchem Paket gehören Klassen, die eine package-Anweisung nicht enthalten?
- Zum Default-Paket
- Zugeständnis an kleine Programme, um den Aufwand mit package- und import-Anweisungen und Unterverzeichnissen zu ersparen.
- Klassen des Default-Pakets können ohne explizite import-Anweisung verwendet werden.

Das public-Attribut

- Sichtbarkeit von Klassen
- Eine Klasse A kann eine andere Klasse B einbinden, wenn eine der folgenden Bedingungen erfüllt ist.
  - Entweder gehören A und B zu dem selben Paket oder
  - die Klasse B wurde als public deklariert.
- Sollen nur Default-Pakete verwendet werden, spielt es keine Rolle, ob eine Klasse vom Typ public ist.
- Anzeige in Eclipse



Zusammenfassung

- Instanzen von Klassen werden in Java zur Laufzeit erzeugt.
- Zugriff auf Objekte erfolgt wie in C++.
- Jede Klasse in Java ist Bestandteil eines Pakets
- Quell-Code-Dateien von Paketen werden in einer Verzeichnisstruktur abgelegt.
- Zugriff auf Objekte eines Pakets kann mit import erfolgen.
- Es gibt für viele Anwendungsklassen Standardpakete in Java.