

Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- Exceptions
- **Java Klassenbibliotheken**
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- Grafik
- Beans
- Integrierte Entwicklungsumgebungen

Klassenbibliotheken

Zu jeder Java-Umgebung (JDK, JRE) gehört eine Menge von **Standardklassen** (Standard API), die in **Paketen organisiert** sind. Den vollständigen Umfang und die vollständige Beschreibung kann man in der API-Dokumentation finden. Hier werden nur die wichtigsten Pakete angegeben.

Das Paket **java.lang** (und damit alle Klassen dieses Pakets) werden vom Compiler automatisch importiert. Will man eine Klasse eines anderen Pakets nutzen, muss man dieses Paket erst **importieren**.

Wichtigsten Pakete der Java Klassenbibliothek

Paket	Beschreibung
<code>java.lang</code>	Grundklassen
<code>java.math</code>	Grosse Zahlen (BigInteger, BigDecimal)
<code>java.util</code>	Verschiedene nützliche Klassen inkl. Collections
<code>java.io</code>	Ein-/Ausgabe
<code>java.net</code>	Netzwerkprogrammierung
<code>java.security</code>	Zugriffskontrolle und Authentifizierung
<code>java.text</code>	Verarbeitung von Text, Datum, Zahlen unabhängig von einer Sprache
<code>java.applet</code>	Applets
<code>java.awt</code>	AWT-Klassen
<code>java.beans</code>	Beans-Komponenten
<code>javax.swing</code>	Swing-Klassen

Bemerkung 1: Es gibt noch Unterpakete, wie z.B. `java.util.jar`.

Bemerkung 2: Vollständige Beschreibung in der API-Dokumentation.

Java Standard Extension API

Extension APIs sind APIs, die nicht zum Kern-API gehören. SUN hat eine Reihe von "Standard Extension APIs" definiert (javax.*) , u.a.:

- **Java Security API**
Aufgrund von Exportbeschränkungen in den USA enthält dieses API Teile des Pakets java.security.
- **Java Media API**
Zum Erstellen von Multimedia-Anwendungen (z.B. Java3D)
- **Java Enterprise API**
JDBC, IDL, CORBA
- **Java Commerce API**
eCommerce

Klassen in java.lang

Object	: schon behandelt
Wrapperklassen	: schon behandelt
ClassLoader	: abstrakte Klasse zum Laden von Klassendateien
Compiler	: (Platzhalter für) JIT-Compiler
Package	: Pakete
Runtime	: Interaktion mit der Systemumgebung
SecurityManager	: Eigene Security Policy definieren
Throwable	: Exceptions (schon behandelt)
Math, StrictMath	: gleich
Class	: gleich
System	: gleich
Process, Thread	: später

Klassen java.lang.Math und java.lang.StrictMath

Unterschied:

java.lang.StrictMath generiert auf allen Systemen ein bitweise exaktes Resultat, java.lang.Math kann dafür evtl. schneller sein.

Beide Pakete haben eine Vielzahl von mathematischen Funktionen als **Klassenmethoden** (abs, sin, cos, exp, log, pow, sqrt, ...)

Beispiel:

```
class Test {
    public static void main(String[] args) {
        double d = StrictMath.pow(StrictMath.PI, 4.0);
        System.out.println(d);           // Ausgabe PI^4
    }
}
```

Klasse java.lang.Class

Klassenobjekte werden nur von der JVM erzeugt, es gibt **keinen expliziten Konstruktor**. Sobald eine Klasse geladen wird, wird ein Class-Objekt dazu erzeugt. Es gibt diverse Funktionen, um Informationen zu einer Klasse zu erhalten.

```
class Test {
    public static void main(String[] args) {
        Test o = new Test();
        System.out.println("Klassenname ist " + o.getClass().getName());

        try {
            Class c = Class.forName("Test");
            c = c.getSuperclass();
            System.out.println("Superklasse von Test ist " + c.getName());
        } catch (Exception e) {
            System.out.println("Fehler");
        }
    }
}
```

Ausgabe:

```
Klassenname ist Test
Superklasse von Test ist java.lang.Object
```

Klasse java.lang.Class

Weiterhin gibt es eine Methode `newInstance()`, die ein Objekt zu einer gegebenen Klasse instanziiert. Über die Nutzung von Klassenvariablen ist man nicht daran gebunden, zur Übersetzungszeit die Klasse wissen zu müssen.

Beispiel:

```
class Test {
    static Object neueInstanz(Object o)
        throws InstantiationException, IllegalAccessException {
        Class c = o.getClass();           // Klasse erfragen
        Object o2 = c.newInstance();      // neue Instanz der Klasse erzeugen
        return o2;                       // neues Objekt zurückgeben
    }

    int x;
    Test() { x = 4711; }

    public static void main(String[] args) {
        Test i = new Test();
        try { System.out.println(i + " " + neueInstanz(i));
        } catch (Exception e) { System.out.println("Fehler"); }
    }
}
```

Ausgabe: test@73d6a5 test@111f71

Klasse java.lang.System

Einige nützliche Methoden und Datenfelder, u.a.

in, out, err	: Standardeingabe, -ausgabe, -fehlerausgabe
currentTimeMillis()	: aktuelle Zeit (z.B. für Zeitmessungen)
exit(status)	: Beenden der aktuellen JVM
gc()	: Garbage Collector
getProperty(name)	: Property erfragen

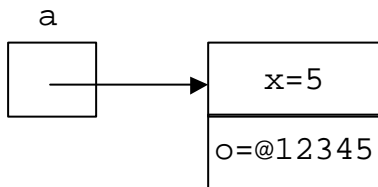
```
class Test {
    public static void main(String[] args) {
        // Startzeit
        long t0 = System.currentTimeMillis();

        // Garbage Collector aufrufen
        System.gc();

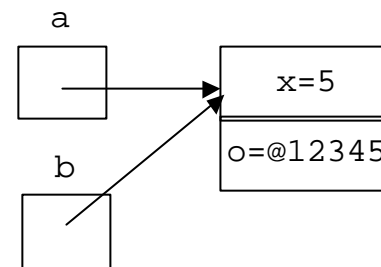
        // Verbrauchte Zeit = jetzige Zeit - Startzeit
        System.out.println("Der Garbage Collector braucht " +
            (System.currentTimeMillis() - t0) + " ms");
    }
}
```

Interface java.lang.Cloneable

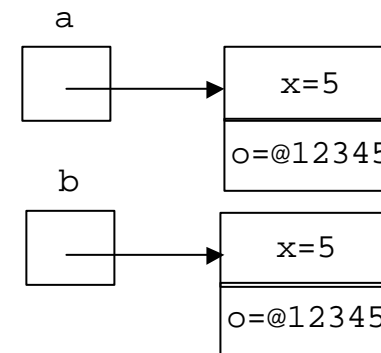
Clonen ist das exakte Kopieren eines bestehenden Objektes, d.h. unter einer neuen Adresse (Referenz) existiert ein Objekt, dessen Inhalt (Datenfelder) exakt dem Ursprungsobjekt entsprechen.



Kopieren $b = a$:



Clonen $b = a.clone()$:



Interface java.lang.Cloneable

Unterstützt eine Klasse das Clonen eines Objektes, so muss sie das **Interface Cloneable implementieren** und ggfs. die **Methode clone() von Object überschreiben**.

```
public interface Cloneable { }    // keine Methoden, keine Datenfelder
```

Ob ein Objekt o "gecloned" werden kann, kann man feststellen mit:

```
if(o instanceof Cloneable) ...
```

Die Methode `clone()` aus `Object` ist ausreichend um **flache Objekte zu kopieren**, d.h. wo die Datenfelder von einfachem Typ sind. Hat aber ein Klasse Datenfelder, die Referenzen auf andere Objekte besitzen (**tiefes Objekt**), so muss eine eigene `clone()`-Methode sicherstellen, dass auch die Referenzen "gecloned" werden, d.h. auf neue Objekte zeigen.

Beispiel

```
class Klasse1 implements Cloneable {
    int x; // Datenfeld einfachen Typs
    Klasse1() { x = 4711; } // Konstruktor

    public Object clone() throws CloneNotSupportedException {
        return super.clone(); // flache Kopie: clone() ausreichend
    }
}

class Klasse2 implements Cloneable {
    int y; // Datenfeld einfachen Typs
    Klasse1 k1; // Datenfeld eines Referenztyps

    Klasse2() { // Konstruktor
        y = 31415;
        k1 = new Klasse1();
    }

    public Object clone() throws CloneNotSupportedException {
        Klasse2 k = (Klasse2) super.clone(); // zuerst mal einfach kopieren
        k.k1 = (Klasse1)k1.clone(); // für k1 ist tiefe Kopie notwendig
    }
}
```

Zeit und Datum

java.util.Date : aktuelle Zeit besorgen (msec seit 1.1.1970)
java.util.DateFormat : Formatieren einer Zeit
java.util.Calendar : Umwandlung in Kalender

```
import java.util.Date;
import java.text.*;

class Test {
    public static void main(String[] args) {
        Date now = new Date(); // aktuelle Zeit besorgen
                               // Formatierer erzeugen mit Default-Regeln

        DateFormat defaultFormat = DateFormat.getDateInstance();
        System.out.println(defaultFormat.format(now));

                               // Formatierer mit eigenem Format erzeugen
        DateFormat meinFormat = new SimpleDateFormat("E yyyy.MM.dd hh:mm:ss");
        System.out.println(meinFormat.format(new Date()));
    }
}
```

Ausgabe:

02.04.2001

Mo 2001.04.02 10:55:41

Zeit und Datum

```
import java.util.*;
class Test {
    public static void main(String[] args) {
        Calendar cal = Calendar.getInstance();           // Kalenderinstanz besorgen

        cal.setTime(new Date());                        // Zeit auf aktuelles Zeit setzen
        System.out.println("Tag im Jahr ist " + cal.get(Calendar.DAY_OF_YEAR));

        // Datum auf den 9.11.1992 setzen
        cal.set(Calendar.YEAR, 1992);
        cal.set(Calendar.MONTH, Calendar.NOVEMBER);
        cal.set(Calendar.DAY_OF_MONTH, 9);
        System.out.println("9.11.1992 war ein " + cal.get(Calendar.DAY_OF_WEEK));

        // Was ist das Datum in 30 Tagen?
        cal.setTime(new Date());
        cal.add(Calendar.DATE, 30);
        System.out.println("30 Tage weiter ist " + cal.getTime());
    }
}
```

Ausgabe:

```
Tag im Jahr ist 92
9.11.1992 war ein 2
30 Tage weiter ist Wed May 02 11:11:14 GMT+02:00 2001
```

java.util.Random: Zufallszahlen

In vielen Bereichen (z.B. Simulationen) benötigt man **(Pseudo-) Zufallszahlen**. Zwei Konstruktoren existieren, einer ohne Parameter und einer mit einem long-Argument, um z.B. gleiche Zufallszahlensequenzen zu bekommen.

```
import java.util.Random;

class Test {
    public static void main(String[] args) {

        // Zufallszahlengenerator erzeugen
        Random r = new Random();

        System.out.print("Die nächsten Lottozahlen sind: ");
        for(int i=0; i<6; i++)
            // Zahl zwischen 1 und 49 erzeugen
            // Hierbei leider Wiederholung möglich (im Gegensatz zu Lotto)
            System.out.print(r.nextInt(49)+1 + " ");
    }
}
```

Ausgabe:

Die nächsten Lottozahlen sind: 20 47 34 38 44 20

java.util.StringTokenizer: String aufspalten

Ein StringTokenizer wird dazu verwendet, einen **String in Teilkomponenten (z.B. Wörter) aufzuspalten**. Dazu kann man eigene Trennzeichen angeben.

```
import java.util.StringTokenizer;
class Test {
    public static void main(String[] args) {
        String str = "Hurra, heute ist wieder Java-Vorlesung";

        // Default-Trennzeichen für Tokenizer
        StringTokenizer st = new StringTokenizer(str);
        while(st.hasMoreTokens())
            System.out.print(st.nextToken() + "\t");
        System.out.println();

        // Jetzt sind Leerzeichen, Komma und Minus Trennzeichen
        st = new StringTokenizer(str, " , -");
        while(st.hasMoreTokens())
            System.out.print(st.nextToken() + "\t");
    }
}
```

Ausgabe:

```
Hurra, heute ist wieder Java-Vorlesung
Hurra heute ist wieder Java Vorlesung
```