# Java Zusammenfassung

موجز جافا
(annett.thuering@informatik.uni-halle.de / 2017)

## Aufbau eines Java-Programms ـ بناء برنامج جافا

```
public class Klassenname {
   public static void main(String[] args) {
         // كود برمجي
   }
}
```

## Compiler مترجم / محوّل برمجي

```
javac Klassenname.java      Bytecode يترجم الأوامر إلى
java Klassenname            Bytecode ينفّذ الـ
```

يجب أن يطابق اسم الكلاس لاسم ملف الجافا :

Klassenname.java = Klassenname

## Kommentare التعليقات

```
// تعليق                    لكتابة التعليقات في سطر واحد
/*  .....  */               لكتابة التعليق في أكثر من سطر
```

## Ausgabe ـ المخرجات

```
System.out.println(Ausgabe);   مع النزول سطر بعد الإخراج
System.out.print(Ausgabe);     بدون النزول سطر بعد الإخراج
```

z.B. Ausgabe: "Text1" + Variable + "Text2"

## Primitive Datentypen ـ متغيرات بدائية

ـ هذه المتغيرات تكون معرفة (موجودة) مسبقا في لغة البرمجة ففيها الحجم والقيَم والعمليات معرفة مسبقا

| الاسم | الحجم (بايت) | مجال القيمة |
|---|---|---|
| byte | 1 (8 Bit) | $-2^7$ - $2^7-1$ |
| short | 2 (16 Bit) | $-2^{15}$ - $2^{15}-1$ |
| int | 4 (32 Bit) | $-2^{31}$ - $2^{31}-1$ |
| long | 8 (64 Bit) | $-2^{63}$ - $2^{63}-1$ |
| float | 4 (32 Bit) | $\pm 3.4*10^{38}$ |
| double | 8 (64 Bit) | $\pm 1,79 * 10^{308}$ |
| char | 2 (16 Bit) | Unicode |
| boolean | 1 (8 Bit) | true, false |

## Automatische Typkonvertierung ـ التحويل التلقائي لأنواع القيم

byte → short → int → long → float → double
char ↗

## Konstanten ـ الثوابت

```
final Datentyp KONSTANTE = Wert;
final int JAHR = 2015;                    مثال
```

## Regeln für Bezeichner ـ قواعد كتابة المعرّفات

ـ أسماء المتغيرات و" المنهج" و أنواع البيانات ـ تسلسل من أحرف وأعداد
ـ يبدأ بـ حرف أو $ ـ اللغة حسّاسة وتفرّق بين الأحرف الانجليزية الكبيرة والصغيرة
ـ لا يمكن كتابة الكلمات المفتاحية كأسماء للمتغيرات وغيرها

## Variablendeklaration الإعلان عن متغيّر

```
Datentyp variablenname;
```

أي نكتب نوع البيانات وبعدها مباشرة اسم المتغير مفصولين بمسافة بينهما

```
int zahl;          الإعلان عن متغير واحد (Deklaration)
int a, b, c;       الإعلان عن أكثر من متغير من النوع نفسه
```

### Deklaration (Definition) mit Initialisieruung ـ الإعلان عن متغير وتهيأته

```
int zahl = 5;
String name = "Rudi";
```

## Eingaben über die Konsole ـ الإدخال عن طريق لوحة التحكّم

### Die Klasse Scanner ـ الفئة / الكلاس :(النوع الأول)

```
import java.util.Scanner;
public class Klassenname {
   public static void main(String[] args) {
     Scanner in = new Scanner(System.in); }}
```

**(Rückgabetyp int)** ـ الإدخال الأعداد الصحيحة

```
z.B. izahl = in.nextInt();
```

**(Rückgabetyp double)** ـ لإدخال الأعداد الكسريّة

```
z.B.  dzahl = in.nextDouble();
```

**(Rückgabetyp String)** ـ لإدخال سلاسل الأحرف(كلمات)

```
z.B. wort = in.next();
```

**(Rückgabetyp String)** ـ لقراءة الأسطر بما فيها المسافات الفارغة

```
z.B. zeile = in.nextLine();
```

### Die Klasse BufferedReader ـ الفئة / الكلاس (النوع الثاني)

```
import java.io.*;
public class Klassenname {
   public static void main(String[] args)
                              throws IOException {
     BufferedReader in = new BufferedReader
          (new InputStreamReader(System.in));
```

**(Rückgabetyp String)** ـ قراءة الأسطر بما فيها المسافات

```
z.B. zeile = in.readLine();
```

**(Rückgabetyp int)** ـ لتحويل الأحرف إلى أعداد

```
z.B. izahl = Integer.parseInt(zeile);
```

**(Rückgabetyp double)** ـ للتحويل من نوع الكلمات إلى نوع الأعداد الكسرية

```
z.B. dzahl = Double.parseDouble(zeile);
```

## Operatoren ـ المُعاملات

## Assoziativitäten (Klammerung) ـ الترابُط

rechtsassoziativ: المُعامِلات التي تستخدم من الاتجاه الأيسر التعبيرات الشرطية , معاملات التخصيص

linksassoziativ: باقي المعاملات

## Prioritäten ـ الأولويّات

| الأولوية | المُعامِل | ملاحظة |
|---|---|---|
| 1 | o.a, i++,a[],f(),. | ـ مُعامِل يستخدم من الاتجاه الأيمن<br>ـ إذا المعامل هنا بعد المتغيّر |
| 2 | -x, !, ~, ++i | ـ مُعامِل يستخدم من الاتجاه الأيسر<br>ـ إذا المعامل هنا قبل المتغيّر |
| 3 | new C(), (type) x | انشاء كائن |
| 4 | *, /, % | الضرب والقسمة وباقي القسمة |
| 5 | +, - | الجمع والطرح |
| 6 | <<, >>, >>> | إزاحة |
| 7 | <, >, <=, >= | أكبر , أصغر و..... |
| 8 | ==, != | يساوي أو لا يساوي |
| 9 | & | الواو المنطقيّة ( و ) |
| 10 | ^ | الـ " بالضبط أو " المنطقيّة ( أو) |
| 11 | \| | الـ " أو " المنطقيّة ( أو ) |
| 12 | && | الـ " و " المنطقيّة ( و ) |
| 13 | \|\| | الـ " أو " المنطقيّة ( أو ) |
| 14 | ?: | تعبير شرطي |
| 15 | =, +=, -=, *=, /= | تحديدات, تخصيصات |

## Kontrollstrukturen ـ هيكلة التحكّم

### ـ أوامر مشروطة : إذا { ... } وإلا { ... } خلاف ذلك

```
if (شروط ) {
     // يتم تنفيذ الأوامرالموجودة هنا في حال كانت الشروط محقّقة
}
else {
       Anweisungen ...  أوامر
     // تم تنفيذ الأوامر الموجودة هنا في حال كانت الشروط السابقة غير محققة
     }
```

## Fallunterscheidung (Auswahl) ـ التمييز بحسب الحالة

```
switch (متغيّر) {
   case 1 قيمة: Anweisungen ... أوامر
     //  break يتم تنفيذ الأموار حتى الكلمة
     //  و ذلك في حال تساوى قيمة 1 مع متغيّر
              break;
   case 2 قيمة: Anweisungen ... أوامر
     //  break يتم تنفيذ الأموار حتى الكلمة
     //  و ذلك في حال تساوى قيمة 2 مع متغيّر
              break;
   ...
   default:  Anweisungen ... أوامر
     // يتم تنفيذ الأوامر المكتوبة هنا في حال لم تساوي أي قيمة سابقة للمتغير
}
```

⚠ ـ يجب أن تكون نوع بيانات المتغير فقط واحد من المتغيرات البدائية الموجودة في لغة الجافا مسبقا

Zum Beispiel: int , double … على سبيل المثال

ـ قيمة 1 و قيمة 2 هم ثوابت

## Wiederholungen ـ حلقات التكرار

### while – Schleife (solange …) ـ طالما

```
while (شرط) {
  //    ـ نفّذ الأوامر المكتوبة بين هذين القوسين طالما أنّ الشرط صحيح
}
```

### do while – Schleife (mindestens einmal aber solange …)

ـ طالما :تنفّذ على الأقل مرّة واحدة في حال كان الشرط غير محقّق

```
do {
          //    ـ نفّذ الأوامر المكتوبة بين القوسين على الأقل مرّة واحدة و ذلك في
          //    حال كان الشرط بين القوسين في الأسفل غير محقّق
} while (شرط );
```

### for – Schleife (Spezialfall: Zählschleife) ـ من أجل

```
for (تهيئة متغيّر ; الشرط ; طول الخطوة) {
  // أوامر
}
```

مثال : سيتمّ هنا تنفيذ الأوامر التي بين القوسين عشر مرات

```
for (int i = 0; i < 10; i++) {
   // أوامر
}
```

### for each – Schleife für Arrays ـ من أجل كل : ( مع المصفوفات)

```
double[] a = new double[10];

for (double x : a) {
   System.out.println(x);
}
```

## vorzeitiges Beenden einer Schleife ـ الإنهاء المبكر للدوّرة

```
continue;
```
ـ استمرار الحلقة

- springt zum Ende des Schleifenrumpfes ـ يقفز إلى نهاية الدوّارة

```
break; (Abbruch )
```
ـ توقّف

ـ اذا تمّت كتابتها في دوّارة سيتمّ ترك الدوّارة ونتفيذ الأوامر التي بعدها مبا شرة

ـ في حال كتبت في " سويتش" سيتم ترك الـ "كيس" الحالية والانتقال للتالية

## Exceptions ـ الاستثناءات

- ArrayIndexOutOfBoundsException (فهرس غير صالح)
- StringIndexOutOfBoundsException ( ترتيب / موقع غير صالح )
- NullPointerException (الوصول إلى مرجع فارغ)
- NumberFormatException (نوع " سترنغ " لا يحتوي على أرقام)
- ArithmeticException (خطأ رياضي . مثل القسمة على الصفر)

- Exceptionbehandlung: ـ معالجة الاستثناءات

```
try {
    // أوامر
}
catch (1 استثناء) {
   // أوامر لمعالجة الاستثناء الأول
}
catch (2 استثناء) {
   // أوامر لمعالجة الاستثناء الثاني
 }
 ...
finally {
// يتم تنفيذ الأوامر المكتوبة هنا على كل الأحوال
}
```

## Methoden überladen ـ الاجراءات المتكررة ـ التحميل المتكرر

ـ اكتظاظ المثود يعني أن يوجد أكثر من ميثود لها نفس الاسم ولكن لكل منها قائمة بلراميتر مختلفة عن الأخرى

1. الخيار الأول: عدد مختلف من البراميترات
2. الخيار الثاني: على الأقل بلراميتر واحد له نوع بيانات مختلف

ـ الهدف منها تطبيق إجراءات متشابهة على أنواع بيانات مختلفة

(Methode System.out.println(…) أنظر )

```
public void println()
public void println(String s)
public void println(int x)
...
```

## Referenztypen ـ أنواع مرجعيّة

## Arrays ـ مصفوفة

### eindimensionale Arrays ـ مصفوفة ذات بُعد واحد

خصائص : ـ العنا صر تكون في الذاكرة خلف بعضها البعض

ـ جميع عنا صر المصفوفة من نفس نوع البيانات

```
int [] zahlenfeld;          // الإعلان عن متغيّر
                            // تعريف متغيّر
zahlenfeld = new int[طول];

// إعلان + تعريف
int [] zahlenfeld = new int[طول];
// إعلان + تعريف + تهيئة
int [] gerade = { 2, 4, 6, 8, 10 };
```

ـ طول المصفوفة Länge des Arrays

```
zahlenfeld.length
```

ـ الو صول لعنصر من المصفوفة Zugriff auf ein Element des Arrays

- Zugriff über Index ـ عن طريق الفهرس

ـ يبدأ فهرس المصفوفة بالـ 0 وينتهي بطول المصفوفة ناقص واحد

- kleinster Index: 0; größter Index: Länge-1

```
zahlenfeld[index] = 7;          مثال
```

### mehrdimensionale Arrays ـ مصفوفة متعدّدة الأبعاد

⚠ ـ في جافا المصفوفة ثنائية الأبعاد هي في الواقع مصفوفة أحاديّة البعد التي تحوي مصفوفات أحادية الأبعاد كعناصر في هذه المصفوفة

ـ عدد أزواج الأقواس يحدّد كميّة أبعاد المصفوفة

```
int [][] matrix;    // الاعلان عن مصفوفة ثنائية الأبعاد
int [][][] quader; // الاعلان عن مصفوفة ثلاثيّة الأبعاد

                    // أمثلة عن تعريف المصفوفات
matrix = new int[5][10];        // ثنائية الأبعاد
Quader = new double[5][10][2];// ثلاثيّة الأبعاد
```

ـ طول المصفوفة Länge des Arrays

```
matrix.length     // طول البُعد الأوّل
matrix[2].length // طول السطر الثاني من البُعد الثاني
```

ـ الو صول لعنصر من المصفوفة Zugriff auf ein Element des Arrays

```
matrix[0][0] = 7;       // بعدين
quader[0][0][0] = 7;   // ثلاثة أبعاد
```

## Kommandozeilenparameter

ـ تتم كتابتها عند بداية البرنامج

ـ يتم التعامل معها/استرجاعها من خلال args الموجودة في الميثود main المصفوفة

```
java Programmname param1 param2

public static void main(String[] args) {
  System.out.println(args[0]);     → param1
```

# Konzepte der Objektorientierten Programmierung

## ـ مفاهيم البرمجة كائنيّة التوجّه

**أولا : تغليف البيانات ودوالّها المُعالجة لها(كلاسات)**

### Aufbau einer Klasse ـ بناء كلاس (فئة)

```
class NamederKlasse {
  // Attribute        الصفات : سواء كانت متغيّرات أو خاصيات
  // Methoden        يأتي بعد الصفات الميثود (طريقة) التي تعالج الصفات
}
```

ـ الكلاس هو نوع بيانات( يُحدّد كيف يجب أن يبدو حفظ كائن معيّن)

**الكلاس ( الفئة ) = قالب لعدّة كائنات متشابهة**

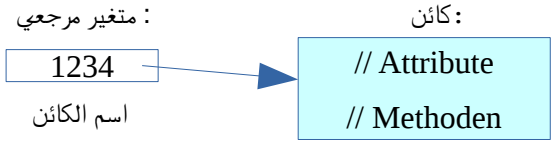
### Objekte / Instanzen einer Klasse ـ كائن / مرجعيّة كلاس

#### Referenzvariable deklarieren ـ الإعلان عن متغيّر مرجعي

```
NamederKlasse objektname;
```

#### Objekt erzeugen und der Refernzvariablen zuweisen:

```
objektname = new NamederKlasse();
```




1234 (عنوان)

المعامل new : لحجز مساحة التخزين في الذاكرة

### Methoden ـ الدوالّ

ـ تستخدم بشكل أساسي للاتصال بالكائن ومعالجته

على سبيل المثال لإعطاء قيَم لكائن معيّن (تغيير حالة الكائن) أو قراءة قيمة معيّنة من الكائن (استعلام حالة الكائن) وبالتالي تحديد تصرّفات الكائن

```
Rückgabetyp Methodenname(Parameterliste){
    // أوامر
    //ـ إرجاع قيمة معيّنة عند الحاجة
    // return Ausdruck;     Ausdruck=تعبير
}
```

لجعل الدّلة لا تُرجع أيّة قيمة نكتب مكان نوع الإرجاع `void`

→ Rückgabetyp = `void` نوع الإرجاع : فراغ

(return) يقيّم التعبير وينهي بعدها الميثود ويرُجع قيمة التعبير

### Zugriff auf Komponenten eines Objektes
### الوصول إلى عناصر كائن ما

- Zugriff erfolgt über den Operator الوصول إلى عناصر كائن ما.

```
Objektname.Attribut
bzw.
Objektname.Methode(Parameterliste)
```

### Die Klasse Object ـ الكلاس أوبجكت (كائن)

ـ هي الكلاس الأساسي / الفئة الأساسية لكل فئات البرنامج

ـ وتتضمّن الفئات الهامّة التالية:

#### Kopie eines Objektes erzeugen ـ انشاء نسخة من كائن

```
protected Object clone()
```

#### Vergleich zweier Objekte ـ المقارنة بين كائنين

```
boolean equals(Object o)

@Override
public boolean equals (Object o) {
   if (o == null) {return false;}
   if (! o instanceof NamederKlasse)
     {return false;}
   if (o == this) {return true;}
   // (cast) تحويل النوع من o إلى
   NamederKlasse oneu = (NamederKlasse) o;

   // oneu مقارنة خاصة مع
   ...
}
```

#### Liefert String-Darstellung eines Objektes

```
String toString()
@Override
public String toString() {
      return "Zeichenkette";
}
```

### Konstruktoren ـ الدوالّ الانشائيّة

ـ هي ميثود خاصّة تستخدم لتهيئة كائن(الصفات وأماكن التخزين)

ـ تستدعى مباشرة عند انشاء الكائن

ـ يمكن أن يحتوي الكلاس على عدة دوال انشائيّة

خصائص الدوال الانشائية:

أولا : ليس لها نوع إرجاع

ثانيا : لها نفس اسم الكلاس

```
// Konstruktor
  NamederKlasse (Parameterliste) {
      ...
  }
```

### Zugriffsrechte ـ امكانيّة الوصول

ـ تكون مباشرة أو غير مباشرة قبل كل متغير أو ميثود

ـ مباشر (private, public)

ـ غير مباشر (package default)

#### private ـ خاص

ـ يسمح بالوصول فقط إلى الكائنات من نفس الكلاس

#### package default ـ مجموعة/حزمة افتراضية

ـ تسمح بالوصول إلى كل الكائنات التي من نفسالمجموعة / الحزمة

#### public ـ عام

ـ تسمح بالوصول إلى كل الكائنات من مجموعة / حزمة اختياريّة

### this : ـ هذا

ـ تحتوي على مرجع يشير إلى الكائن نفسه بحيث يمكن استخدام هذه الكلمة المفتاحية للوصول لكافة مكونات الكائن

```
this.Attribut       bzw.
this.Methode(Parameterliste)
```

### Klassenattribute/Klassenmethoden (static)
### متغيّرات خاصة بالكلاس / دوالّ خاصة بالكلاس

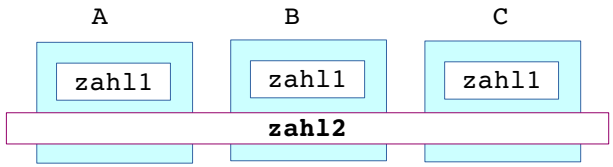
ـ توجد مرة واحدة لكل كائنات الكلاس

ـ توجد فقط عندما لا يكون هناك كائن واحد في الكلاس

⚠ ـ تستطيع الـ `static`-Methoden الوصول فقط إلى `static`- Komponenten

```
class NamederKlasse {
   int zahl1;
   static int zahl2;
        ...
}

A = new NamederKlasse();
B = new NamederKlasse();
C = new NamederKlasse();
```




ـ إمكانيّة الوصول إلى ال Klassenattribut المشترك

```
A.zahl2      B.zahl2
C.zahl2      NamederKlasse.zahl2
```

## Vererbung / Erweitern الوراثة

**المفهوم الثاني: اسناد الدوال والمتغيرات إلى كلاس آخر**

### Zugriffsrechte – Fortsetzung ـ امتيازات الوصول ـ تكملة

`protected` ـ محمي

ـ تسمح بالوصول إلى كائنات من نفس الحزمة " باكيت " وإلى فئات فرعيّة من حزم أخرى

### Basisklasse (Oberklasse) ـ الفئة الرئيسية

```
class BasisKlasse {
```

عناصر private

عناصر protected

عناصر package default

عناصر public

```
}
```

### Subklasse (Unterklasse) ـ الفئة الفرعيّة

ـ الفئات الفرعية توسّع الفئات الرئيسية لإجراءات(ميثود) وثوابت إضافية

```
class SubKlasse extends BasisKlasse {
```

ـ إجراءات(ميثود) وثوابت إضافية ـ

ـ الوصول إلى عناصر الفئة الرئيسية

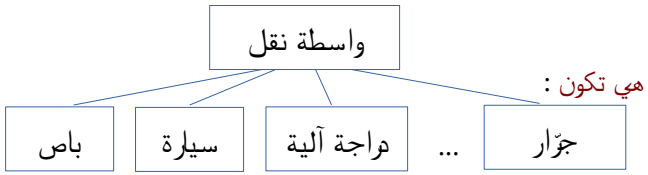
عناصر protected

عناصر package default

عناصر public

```
}
```

### Beispiel: Realisierung von Hierarchien ـ مثال: تحقيق هيكلي




ـ الباص هو واسطة نقل والسيارة هي واسطة نقل فهما متشابهان من حيث كونهما واسطة نقل

ـ كل مرجعية من الفئة الفرعية هي أيضا مثيل من الفئة الأساسية

```
Fahrzeug F1 = new Auto();  //
```
تعددية الأشكال

### Konstruktoren ـ الدوالّ الانشائية

ـ الدوال الانشائية في الفئة الأساسية تكون غير مرئية في الدوال الفرعية على الرغم أنها موجودة وأيضا امتيازات وصولها هي من نوع `public`

ـ دائما ما يتم استدعاء الدوال الدوال الانشائية الموجودة في الفئة الرئيسية (بالتتابع كما في التدرّج الهرمي لهذه الفئات)

ـ التسلسل : تستدعى الدوال الانشائية الخاصة بالفئة الرئيسية أولا وبعدها هلى طول التدرج الهرمي للفئات

ـ نداء الدالة الانشائية الخاصة بالفئة الرئيسية يحدث عن طريق

```
super(Parameterliste);
```

ـ نداء الدالة الانشائية الخاصة بالفئة الرئيسية يكون دائما الأمر الأول في الدالة الانشائية الخاصة بالفئة الفرعيّة

### Referenz super ـ المرجع

ـ في كائن ذو فئة فرعية يسمح هذا المرجع بالوصول إلى العناصر الموروثة من الفئة الرئيسية الأصلية مباشرة

ـ وبالتالي تسمح بالوصول إلى دوال انشائية والإجراءات المتكررة والمتغيرات المحجوبة

### verdeckte Attribute ـ المتغيرات المحجوبة

ـ الأتريبوت الخاصة بالفئة الفرعية التي تحتوي نفس الاسم كالأتريبوت الخاصة بالفئة الرئيسية تغطي في دوال الفئة الفرعية الأتريبوت الخاصة بالفئة الرئيسية

```
class B {
   public int a;
   public B() {a = 5;}
}
 class S extends B {
   public int a;
   public int basis_a;
   public S() {
      a = 10;
      basis_a = super.a;}
}

S testS = new S();
System.out.println(testS.a);          → 10
System.out.println(testS.basis_a); → 5
```

**// Polymorphie**

```
B testBS = new S();
System.out.println("\n " + testBS.a);→ 5
```

### final - Modifikator ـ المعدّل

- `final` الدوالّ المعرّفة لا يمكن تكرارها في الفئات المنتجة
- `final` الفئات المعرّفة لا يمكن وراثتها

### Überschreiben von Methoden ـ تكرار الدوالّ

ـ تهيئة الدالّة بحسب متطلّبات الكلاس الفرعي

ـ الدالة في الفئة الفرعي يجب أن تحتوي على نفس الاسم الخاص بالدالة الموجودة بالفئة الأساسية ونفس البراميترات وأيضا نوع بيانات الإرجاع يجب أن يكون ملائم

ـ الدوالّ المتكرّرة الخاصة بالفئة الأساسية تكون متاحة فقط للفئات الفرعيّة وذلك حصرا عبر super



```
class BM {
   public int zb;
   public BM() {zb = 2;}
   @Override public String toString () {
      return "zb = " + zb; }
}
class SM extends BM {
   public int zs;
   public SM() {zs = 20;}
   @Override public String toString () {
      return super.toString() +
             "\n zs = " + zs; }
}

SM testSM = new SM();
System.out.println(testSM);
```

zb = 2

zs =

**// Polymorphie تعددية الأشكال**

```
20
BM testBMSM = new SM();
System.out.println(testBMSM);
```

### abstrakte Basisklassen ـ فئات مجرّدة

ـ تعرّف الصفات الأساسية في كلاس أساسي

ـ لا يمكن انشاء كائنات من الفئات المجرّدة

ـ يمكن أن تحتوي الفئات المجرّدة على دوالّ مجرّدة التي تكون بدون تنفيذ

```
abstract class Abasis {
   int z;
   abstract public void print (int a);
   public int get_z () {return z;}
}
```

ـ الفئات الحاوية على دوالّ مجرّدة يجب بالضرورة أن تكون معرّفة على أنها مجرّدة

ـ الدوالّ المجرّدة لا يمكن أن تكون من نوع private

## generische Klassen ـ الفئات العامّة

ـ نموذج / قالب من أجل مجموعة من الكلاسات
ـ الفئات من مجموعة معينة يختلفون فقط في نوع بيانات بعض المتغيرات

```
class A {
  int att;
  ...
}
```

```
class B {
  String att;
  ...
}
```

```
class C {
  Person att;
  ...
}
```

Instanz erzeugen:

```
A aobj =        B bobj =        C cobj =
new A();        new B();        new C();
```

ـ في الفئات العامة يتم إضافة عنصر نائب من أجل نوع بيانات في مكان معيّن

```
class<Platzhalter> ABCgen  {
   Platzhalter att;
   ...
   public Platzhalter getAtt () {return att;}
   public void setAtt (Platzhalter w) {
        att = w; }
   @Override public boolean equals(Object o){
       if(o instanceof ABCgen<?>) {
          ABCgen<?> p = (ABCgen<?>) o;
       ...
   }
}
```

ـ الكلاسات العامة يمكن أن توفّر وظائف محددة لمتغيرات عامّة
ـ عند انتاج مرجعية من فئة عامة يجب أن يشار إلى نوع البيانات التي من أجلها يكون العنصر النائب
ـ نوع البيانات يجب أن يكون نفسه نوع مرجعي

```
ABCgen<Integer> aobj =new ABCgen<Integer>();
ABCgen<String> bobj = new ABCgen<String>();
ABCgen<Person> cobj = new ABCgen<Person>();
```

## ArrayList ـ قوائم المصفوفات

ـ تحتفظ بالعناصر في مصفوفة داخلية
ـ بالمقارنة مع المصفوفات العادية طول قوائم المصفوفات يمكن تغييره

```
import java.util.ArrayList;
...
ArrayList<Datentyp> aliste;
aliste = new ArrayList<Datentyp>();
```

## Methoden der Klasse(ArrayList) ـ الميثود الخاصة بالكلاس

Element an einer bestimmten Position ausgeben
`aliste.get(Position)`

Element am Ende der Liste einfügen ـ إضافة عنصر في نهاية القائمة
`aliste.add(Element)`

Element mit Positionsangabe einfügen ـ إضافة عنصر في موقع معيّن
`aliste.add(Position, Element)`

Alle Elemente einer anderen Liste einfügen ـ إضافة عناصر من قائمة أخرى
`aliste.addAll(Position, liste2)`

Element an Position verändern ـ تغيير عنصر في مكان معيّن
`aliste.set(Position, Element)`

Element löschen ـ حذف عنصر / Liste leeren ـ إفراغ قائمة
`aliste.remove(Element)/ aliste.clear()`

Element bestimmter Position löschen ـ حذف عنصر ذو موقع معيّن
`aliste.remove(Position)`

Gibt Größe des Arrays zurück ـ يُرجع طول المصفوفة
`aliste.size()`

Prüft, ob bestimmtes Element enthalten ist (liefert true, false)
ـ يختبر فيما إذا عناصر معيّنة مُحتواة
`aliste.contains(Element)`

Liefert den Index eines Elementes ـ يرجع فهرس عنصر معيّن
`aliste.indexOf(Element)`

## Interfaces / Schnittstellen ـ واجهة بينية

ـ الواجهة البينيّة تفرض على فئة معيّنة تنفيذ دوالّ معيّنة
ـ الدوالّ المكتوبة في الواجهات البينيّة تكون تلقائيا من الأنواع التالية وذلك بدون كتابة الكلمات المفتاحيّة التالية `public, abstract`
ـ الدوالّ لا يمكن أن تكون من نوع `static`
ـ الواجهات البينيّة يمكن أن ترث واجهات بينيّة أخرى
ـ الأتريبوت المعلنة تكون ضمنيّا `public, static, final` ويجب أن يتم تهيأتها
ـ عن طريق الواجهات البينيّة تكون الوراثة المتعدّدة ممكنة
ـ الفئات الفرعية يمكنها أن ترث فئة رئيسية فقط لكنها يمكن أن تنفّذ أكثر من واجهة بينيّة

Beispiel für generisches Interface: ـ مثال على واجهة بينيّة عامّة
Vergleich zweier Objekte: المقارنة بين كائنين

```
interface Comparable<T> {
   int compareTo (T obj2);
}
```

Rückgabewert: 0 bei Gleichheit
< 0 falls Objekt selbst kleiner als `obj2`
> 0 falls Objekt selbst größer als `obj2`

```
class Kl implements Comparable<Kl>{
  private int wert;
 ...
  @Override
  public int compareTo(Kl obj2) {
        ...
  }
}
```

الاستخدام : على سبيل المثال للترتيب

```
import java.util.Collections;
...
ArrayList<Kl> klliste;
klliste = new ArrayList<Kl>();
...
Collections.sort(klliste);
...
```

## Klassen für Zeichenketten ـ الفئة لـ سلسلات الأحرف

### String ـ سلسلة

⚠ **المضمون لا يمكن أن يتغيّر بعد التعريف**
الدوال التالية تنشئ سلسلة جديدة ولا تغير السلسلة نفسها
`replace, toLowerCase`

```
String satz = "Viel Spaß!";
```

Zugriff auf ein Zeichen ـ الوصول إلى رمز / Länge des Strings ـ طول سلسلة
`char charAt(int index) / int length()`

Liefert 1. Position, von ch (Zeichen) / st (String) im String
ـ تمدّ المكان الأول من الحروف المطلوبة
`int indexOf(int ch) bzw. (String st)`

`true`, falls die Zeichenkette `st` im String enthalten ist
ـ ترجع "صحيح" إذا كانت السلسلة تحتوي على st
`boolean contains(String st)`

Vergleich zweier Strings; `true`, falls beide Strings gleich
`boolean equals (Object anObject)`

Liefert Teilstring (ohne das Zeichen an Position `end`)
`String substring(int begin, int end)`

Ersetzen eines Zeichens ـ تبدل رموز معينة مكان أخرى
`String replace(char old, char new)`

Umwandeln in Kleinbuchstaben / Großbuchstaben
ـ تبدل حروف كبيرة إلى صغيرة أو حروف صغيرة إلى كبيرة
`String toLowerCase() / toUpperCase()`

### Stringbuilder und StringBuffer

- veränderbare Zeichenketten

```
import java.lang.StringBuilder;
```

```
...
StringBuilder satz;
satz = new StringBuilder("Viel Spaß!");
```