# Bezeichner

## Erlaubt:

Klein- und Großbuchstaben (a … z, A … Z)  
Viele weitere Zeichen aus dem Unicode-Zeichensatz  
Beispiele: Unterstrich \_, Währungszeichen (€, $, etc.), griechische Buchstaben, etc.

## Nicht erlaubt:

Java-Ziffern am Anfang des Bezeichner (z.B. 4you)  
Sonderzeichen wie ! ?  
Leerzeichen im Bezeichner  
Reservierte Schlüsselwörter wie null, class, ...

# Datentypen

|  |  |  |  |
| --- | --- | --- | --- |
| Bool | 8 | 0 | 1 |
| char | 16 | 0 | 65536 |
| Byte | 8 | -128 | 127 |
| Short | 16 | -32768 | 32767 |
| Int | 32 | -2147483648 | 2147483647 |
| Long | 64 | <-1018 | >2018 |
| Float | 32 | <-1038 | >1038 |
| Double | 64 | <-10308 | >10308 |

# Operatoren

|  |  |
| --- | --- |
| **Operator** | **Priorität** |
| () | hoch |
| ~ ! -- ++ + - cast (unär) |  |
| \* / % |  |
| + - |  |
| <<, >>, >>> |  |
| < ,<=, >, >=, instanceof |  |
| == != |  |
| & |  |
| ^ |  |
| | |  |
| && |  |
| || |  |
| Bedingter ausdruck |  |
| = += -= \*= /= %= | niedrig |

# Sprachelemente

**Zuweisung**: Variable = Ausdruck;

**Block**: { Anweisung ... Anweisung }

**Alternative**: if (Bedingung) { Anweisung } else { Anweisung }

**Mehrfachauswahl**:

switch (Ausdruck) {  
case v1: Anweisung; break;  
case v2: ...  
default: Anweisung; }

**Schleifen**:

for ( Initialisierung; Bedingung; Fortschaltung)  
{  
Schleifenrumpf;  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

while (Bedingung) {Anweisung};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

do { Anweisung } while (Bedingung);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for (element : sammlung) { anweisung }; //Sammlung darf dabei nicht verändert werden

**Objekt anlegen:**

Klassenname Objektname = new Konstruktormethode(Parameter);

# Struktur



# UML-Diagramme

|  |  |
| --- | --- |
| - | Private |
| + | Public |
| ---> | Aggregation  (Klasse wird in anderer Klasse verwendet) |
| <<create>> | Konstruktor |



# Zugriffsrechte

|  |  |  |
| --- | --- | --- |
|  | Im gleichen Package | Klasse Methode |
| Protected | Im gleichen Package, | Klasse, Methode |
| Private | In der klasse | Methode |
| Public | Überall | Klasse, Methode |

# Klassen

**Einbinden:** import java. …

## Wrapper Klassen

Klassen zum Verpacken der Standart-Datentypen

byte Byte  
short Short  
int Integer  
long Long  
float Float  
double Double  
boolean Boolean  
char Character

## String:

public int length() – Länge der Zeichenkette  
public boolean equals(String a) – gibt an ob parameter dem stringobjekt entspricht  
public string replace(String a, String b) – erzeugt neuen string in dem jeder string a durch b ersetzt wird

## Liste: import java.util.\*

**Eigenschaften**: geordnet, unsortiert,Duplikate erlaubt  
**Unterklassen**: ArrayList(Array Repräsentation) und LinkedList()(schnelle Zugriffe auf 1./letztes Element)

**Deklaration**: ArrayList<String> liste  
**Konstruktor**: liste = new ArrayList<String>(); **oder** buchsammlung = new ArrayList<>();

**Methoden**:

Element hinzufügen: liste.add("Kaffee");  
liste.add(idx, "Tee");

Element entfernen: liste.remove(idx);

Länge der Liste: int length = liste.size();

Zugriff auf das Element mit Index idx: String str = liste.get(idx);

## Set: import java.util.\*

**Eigenschaften**: keine Duplikate  
**Unterklassen**: HashSet (ungeordnet, unsortiert)TreeSet (in aufsteigender Reihenfolge sortiert)

## Map: import java.util.\*

**Eigenschaften**: Jeder Eintrag 2 Komponenten, Key-Value-Pair (Schlüssel-Wert-Paar), Zugriff auf den Wert über das Schlüsselobjeckt, Schlüssel ist eindeutig

**Unterklassen**: HashMap (ungeordnet, unsortiert, Duplikate nicht als Schlüssel, aber als Wert erlaubt)  
TreeMap (ungeordnet, sortiert nach Schlüssel, Duplikate nicht als Schlüssel, aber als Wert erlaubt)

**Deklaration**: HashMap<String,String> map;  
**Konstruktor**: map = new HashMap<String,String>();

**Methoden**:

Eintrag (Key-Value-Pair) hinzufügen: map.put(key, value);

Zugriff auf Eintrag mit Schlüssel key: String value = map.get(key);

Eintrag mit Schlüssel key entfernen: map.remove(key);

Anzahl der Einträge: int number = map.size();

Existenz von Wert oder Schlüssel überprüfen: boolean existsKey = map.containsKey(key);  
boolean existsValue = map.containsValue(value);

## Iteratoren import java.util.\*

Deklaration:  
Iterator it = Sammlung.iterator();

Iterator<String> it = list.iterator(); //Iterator auf 1. Element in Liste  
~~Iterator<String> it = array.iterator(); //Iterator auf 1. Element in Array~~

Einheitliche Verarbeitung (Zugriff auf Array und ArrayListe mit gleichen Methoden):

Zugriff auf nächstes Element: it.next();  
Gibt es ein nächstes Element: it.hasNext();  
Aktuelles Element entfernen: it.remove();

## Random import java.util.\*

**Konstruktoren**: Random(), Random(long seed)

**Methoden:**

nextInt(): ganzzahlige Zufallszahl aus dem Wertebereich der int-Werte  
nextInt(int n): ganzzahlige Zufallszahl im Intervall [0, n[  
nextDouble(): Zufallszahl vom Typ double im Intervall [0, 1[

Eingabe:

Scanner eingabe = new Scanner(System.in);  
double nettoBetrag = eingabe.nextDouble();

Ausgabe:

System.out.print("Hello World\n");  
System.out.println("1. Zeile\n\n3. Zeile");

# Objektsammlungen

Arrays – feste LängeDeklaration: String [] arr;  
Alternative: String arr[];

Konstruktor (erzeugt noch keine Zeichenketten!): arr = new String[3];  
Alternative: arr = {"Aufräumen", "Fegen", "Schokolade kaufen"}

Längenattribut : arr.length;

Funktionen/Methoden:

Sondierend: Information über den Objektzustand zurückgeben, keine Veränderung des Objektes  
Verändernd: Objektzustand ändern

**Überladen**: Mehrere Konstruktoren/Methoden mit gleichem Namen und verschiedenen Parametern

# Variablen

**Objektvariablen**: Gültigkeit innerhalb des gesamten Objektes

**Lokale** **Variablen**: Gültigkeit nur innerhalb einer Methode

**Schlüsselwörter:**   
final: schützt Variablen vor Veränderung (Namenskonvention CapsLock)  
static: objektunabhängige Variable

**Statische Datentypen**

**Dynamische Datentypen**

# Static

Nur eine Instanz in einem gemeinsamen Speicher

**Methoden:**Statische Methoden können NICHT auf Objektmethoden zugreifen  
Objektmethoden können auf statische Methoden zugreifen

# Vererbung

Konstruktoraufruf implizit(keine) oder explizit  
Zugriff mit „super“ Operator an erster Stelle (explizit)

**Einbinden:** Schlüsselwort extends

**Statischer Typ:** deklarierten Typ einer Variablen (vom Compiler geprüft)  
**Dynamischer Typ:** Typ des Objekts auf das eine Variable zeigt

## Polymorphie

Objektvariablen:  
können auf Objekte des deklarierten Typs, aber auch auf Objekte aller Subtypen verweisen

Methoden:  
Aufruf kann zur Ausführung unterschiedlicher Methoden zu unterschiedlichen Zeitpunkten führen

## Abstract

**Klassen:**  
Anlegen von Objekten nicht möglich

abstract class Klassenname{}

**Methoden:**  
müssen in einer Kindklasse implementiert werden  
Abstrakte Methoden machen auch die Klasse abstrakt

abstract ReturnType Methodenname(Parameter);

## Interface

public interface Actor

**Einbinden:** Schlüsselwort implements

**Methodendeklaration:**  ReturnType methodenname(Parameter);

## Cast

Explizite Typangabe (Vererbungshierachie runter)  
Implizite Typumwandlung durch Zuweisung(Vererbungshierachie hoch)   
Prüfung während der Laufzeit

## Mehrfachvererbung

Eine Klasse erbt von mehreren Oberklassen.

Java verbietet es für Klassen.  
Java erlaubt es für Interfaces.

# Exceptions

Methode wird sofort ohne Rückgabewert beendet. Client kann Exception auffangen.

**Geprüfte Exceptions**:  
Unterklasse von Exception  
Für erwartbare Fehler, bei denen eine Korrektur möglich ist.  
Müssen gefangen werden (Schlüsselwort throws nach Methodeparameter)

**Ungeprüfte Exceptions:**Unterklasse von RuntimeExceptionFür unerwartete Fehler, bei denen wahrscheinlich keine Korrekturmöglich ist.  
Ungefangene Exceptions führen zu Programmabbruch

Exception-Objekt erzeugen: new ExceptionType("...");

Exception-Objekt an aufrufende Methode übergeben: throw …

Exception-Objekt auffangen: try{Methode}   
catch(ExceptionType){}   
catch(ExceptionType){}  
…  
finally{Wird IMMER aufgeführt}

Java Plattformen

Java Platform, Standard Edition (Java SE), insbes. für Anwendungen auf Standard-PCs  
Java Platform, Enterprise Edition (Java EE), insbesondere für Firmenanwendungen, basiert auf Java SE  
Java Platform, Micro Edition (Java ME), insbesondere für Handys, PDAs  
Java Card Platform (Java Card), für SmartCard-Anwendungen

**package** Klausurbeispiel;

**import** java.util.\*;

**public** **class** Klassenname {

**private** **static** **int** *ersteVariable*; //Objektunabhängige Klassenvariable

**protected** **int** zweiteVariable; //Variable für Package

**public** **int** dritteVariable; //öffentliche Variable

**static** Random *beispielZahlengenerator* = **new** Random();

**public** Klassenname(**int** beispielParameter) { //Konstruktor

*ersteVariable* = beispielParameter;

zweiteVariable = beispielParameter;

dritteVariable = beispielParameter;

}

**public** **static** **void** main(String[] args) {

String erstesArray[] = **new** String[5]; ArrayList <String> ersteArrayList = **new** ArrayList();

**for**(String str : args) {

ersteArrayList.add(str);

}

System.***out***.println(ersteArrayList.get(*beispielZahlengenerator*.nextInt()%ersteArrayList.size()));

}

**private** **int** beispielMethode() {

**int** returnVariable = 0;

returnVariable = *beispielZahlengenerator*.nextInt();

**return** returnVariable;

}

//abstract void abstracteMethode(int beispielparameter);

}