Java
Grundkurs für Programmieranfänger

Dr. Cora Burger

Überblick

1. Grundlagen: Algorithmus, Syntax, Compiler, Laufzeitumgebung
2. Programmiersprache Java
   - Historie, Versionen, Einsatzgebiete
   - Charakteristika , Compiler, JVM, Garbagecollection
   - Installation des SDK, Entwicklungsumgebung
3. Programmierung
   - Programmstruktur
   - Datentypen, Operatoren
   - Bedingungen, Verzweigungen, Schleifen
   - Fehlerbehandlung
4. Objektorientierung
   - Klassen, Eigenschaften, Methoden
   - Vererbung
   - Abstrakte Klassen, Interfaces
   - Polymorphie
5. Framework
   - Überblick über API
   - Streams, Datenstrukturen
6. Erstellung einer Beispielanwendung
   - Swing
   - Ereignissteuerung

Tipps und Tricks

1. Organisatorisches

Zeiten: 30.05.-03.06.2022 09:00-16:00 Uhr

Unterrichtsmaterialien: Folien, Übungsaufgaben

Sonstiges:

Frühstücks-, Mittags-, Nachmittagspause

Bestellung des Mittagessens bis 11 Uhr

Ihr Fokus:

1. Grundlagen: Algorithmus/Syntax

Algorithmus:

- Handlungsvorschriften für eine Problemlösung bestehend aus endlich vielen, wohldefinierten Schritten (vergleichbar mit Back-/Kochrezept)
- Überführung einer Anfangssituation ("Eingabe") in eine Endsituation ("Ausgabe")
- Flexibel an vergleichbare Problemstellungen anpassbar durch Einsatz von Variablen und Verzweigungen
- Ausführbar durch Menschen oder Computerprogramm
- Übersichtliche Darstellung z. B. als Nassi-Shneiderman-Diagramm

Beispiel: Bestimmung des Maximums zweier Zahlen

Syntax:

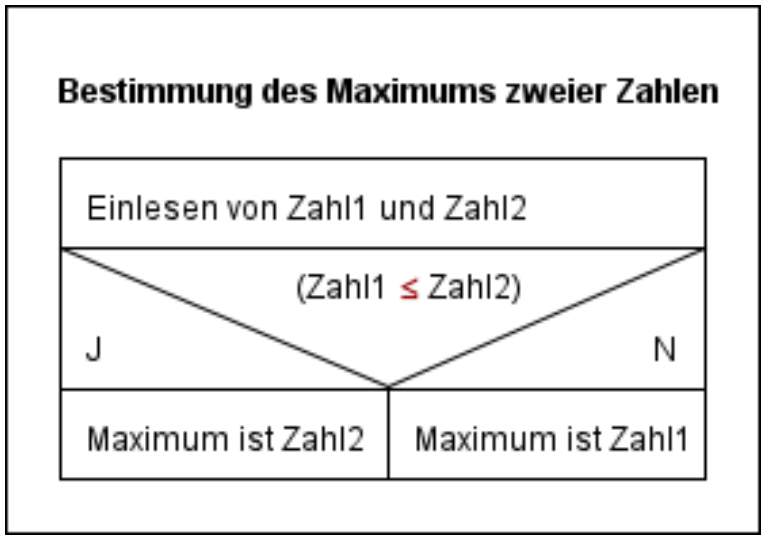
Regelsystem zur Kombination von Zeichen/Wörtern

Beispiel: Grammatik einer (Programmier-)Sprache

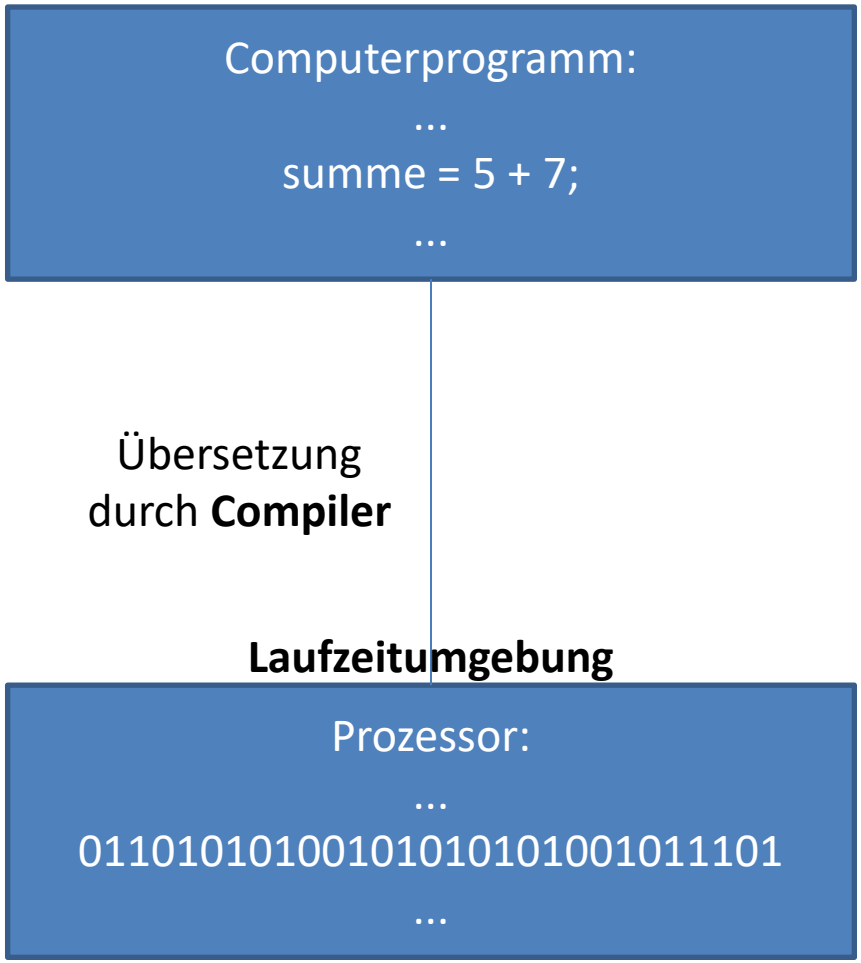
Benötigt für Formulierung der wohldefinierten Schritte

Nassi-Shneiderman-Diagramm

help.structorizer.fisch.lu/index.php?menu=46



Ausführung eines Computerprogramms



2.1 Programmiersprache Java

Historie und Versionen

- 1991 Entwicklungsstart (James Gosling, Sun)
- 1995 Java-Applets
- GNU General Public License
- 2010 von Oracle übernommen,
  April 2019 Lizenzänderung bei Oracle
- Quelloffener Teil als openJDK fortgeführt
  (openjdk.java.net für Version 6-8 für Linux,
  jdk.java.net für Version 7-18)
- Dokumentationen von Oracle:
  Version 8 , 9, 10; 11, 12, 13, 14, 15, 16, 17, **18** Aktuelle stabile Version
- Versionsunterschiede:
  - Neue/veraltete Möglichkeiten
  - Strukturierung

Important Oracle JDK License Update

The Oracle JDK License has changed for releases starting April 16, 2019.

The new Oracle Technology Network License Agreement for Oracle Java SE is substantially different from prior Oracle JDK licenses. The new license permits certain uses, such as personal use and development use, at no cost -- but other uses authorized under prior Oracle JDK licenses may no longer be available. Please review the terms carefully before downloading and using this product. An FAQ is available here.

Commercial license and support is available with a low cost Java SE Subscription.

Oracle also provides the latest OpenJDK release under the open source GPL License at jdk.java.net.

Einsatzgebiete

Viele unterschiedliche Bereiche, u. a.

- Server, Webanwendungen
- Autos
- Mobile Geräte: Tablet, Smartphone
- Elektronikgeräte: Z. B. Waschmaschine

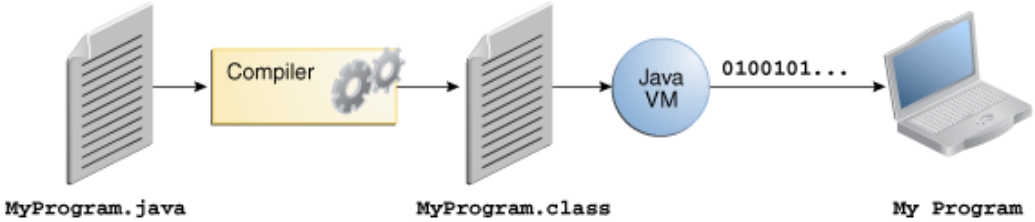
Große Beliebtheit (vgl. www.tiobe.com/tiobe-index)

2.2 Charakteristika

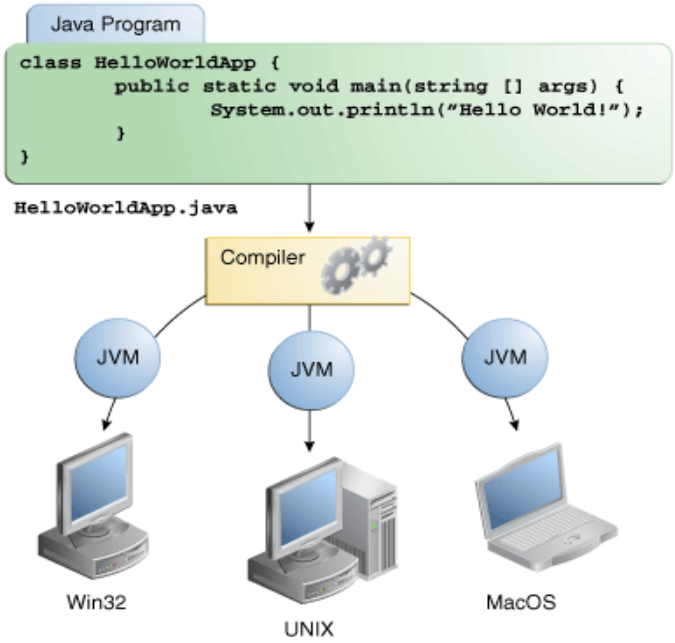
- Objektorientierte Programmiersprache
- Strenge Typisierung
- Kompiliert und interpretiert
- Bestandteile:
  - Programmiersprache
  - **Java Development Kit** (**jdk**): Für Programmentwicklung erforderlich
    Kompiler (Bytecode) + Java Virtual Machine (JVM)
    Java Runtime Environment (**jre**):
    Nur Laufzeitumgebung JVM
  - API (Application Programming Interface

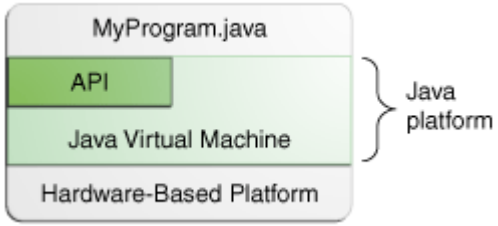
Java Virtual Machine (JVM)

docs.oracle.com/javase/tutorial/getStarted/intro/definition.html



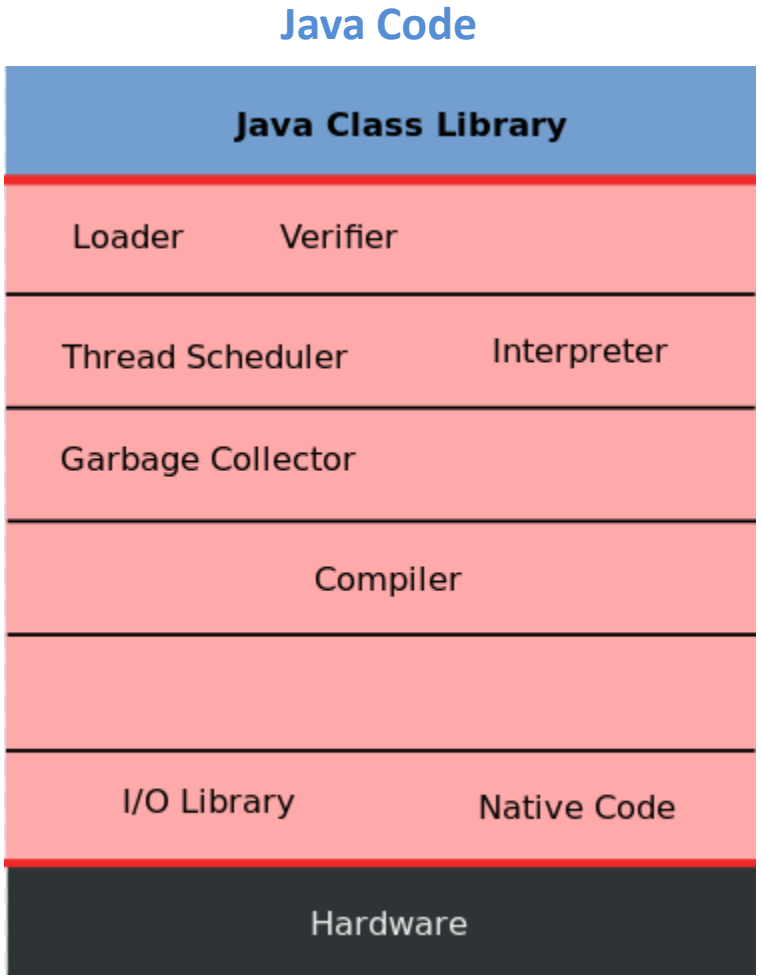
„Plattformunabhängigkeit“





JVM: Bestandteile

- Klassenlader
- Garbage Collection:
  Automatische Speicherbereinigung
- Ausführungseinheit für Java-Threads



2.3 Java-Entwicklung

1. jdk herunter laden und installieren:
   jdk.java.net/18 oder www.oracle.com/technetwork/java/javase/downloads
   newsletter.codejava.net/java-se/install-openjdk-18-on-windows

Einfache Vorgehensweise

2. Texteditor: Z. B. notepad++ 

3. **Kompilieren**:
   `javac ProgrammName.java`
   → `ProgrammName.class` (Bytecode)
   Bündelung mehrerer Programme: `Buendel.jar`
4. Ausführen (in **JVM**):
   `java ProgrammName`
   `java –jar Buendel.jar`

Entwicklungsumgebungen

Vorgehen

2. Explorer/Texteditor
   - Syntax-Highlighting
   - Autovervollständigung
   - Suchen/Ersetzen/Umbenennen (refactoring)

3. + 4. Laufzeit-/Debugumgebung

Beispiele

- www.eclipse.org
- netbeans.org
- www.jetbrains.com/idea: Ultimate, Community Edition

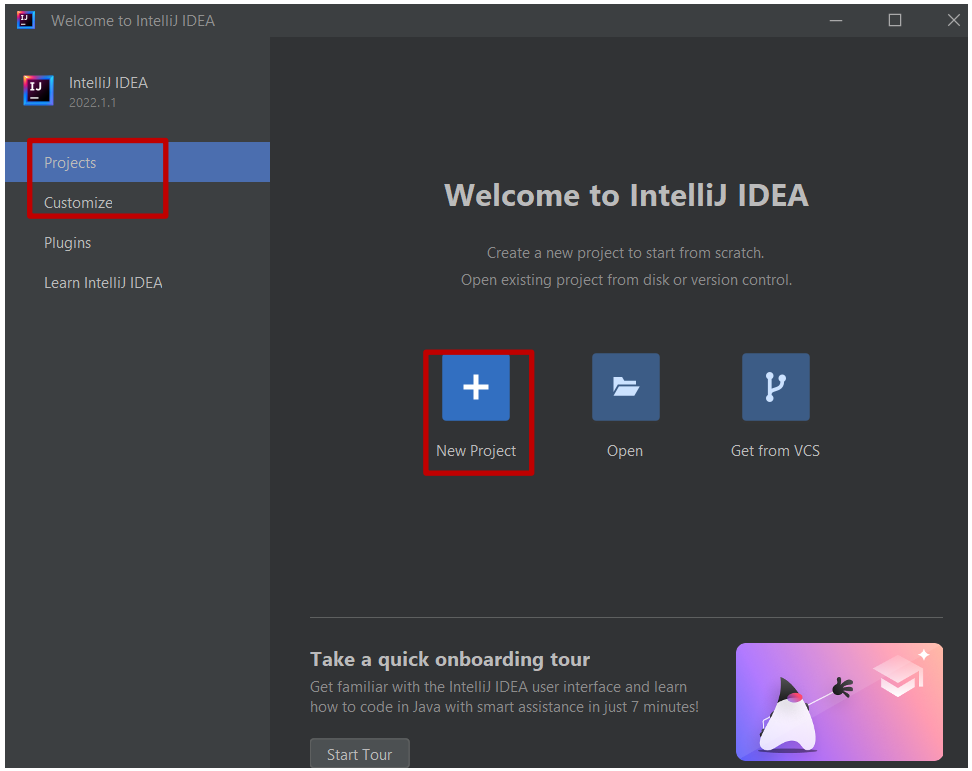
IntelliJ Idea

Herunterladen und installieren von
www.jetbrains.com/idea/download/?fromIDE=#section=windows

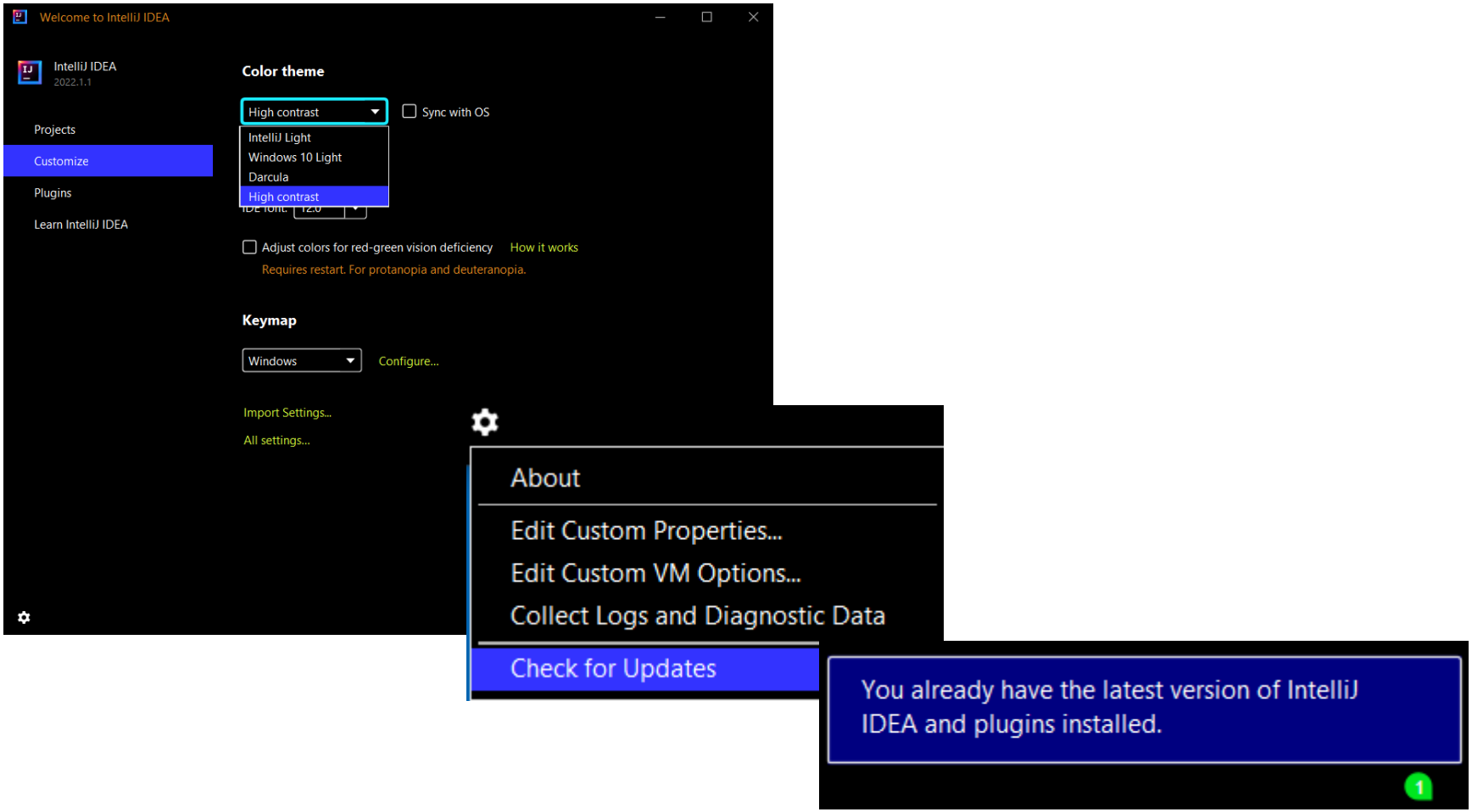
Starten

Konfiguration

Projekte erstellen



Konfiguration / Update



3.1 Programmstruktur

Applikation – **Project**

Weitere Aufteilung:

package – Thematische Zusammenfassung mehrerer Classes

Class:

Basis der Objektorientierung
enthält **Variable** und **Methoden**

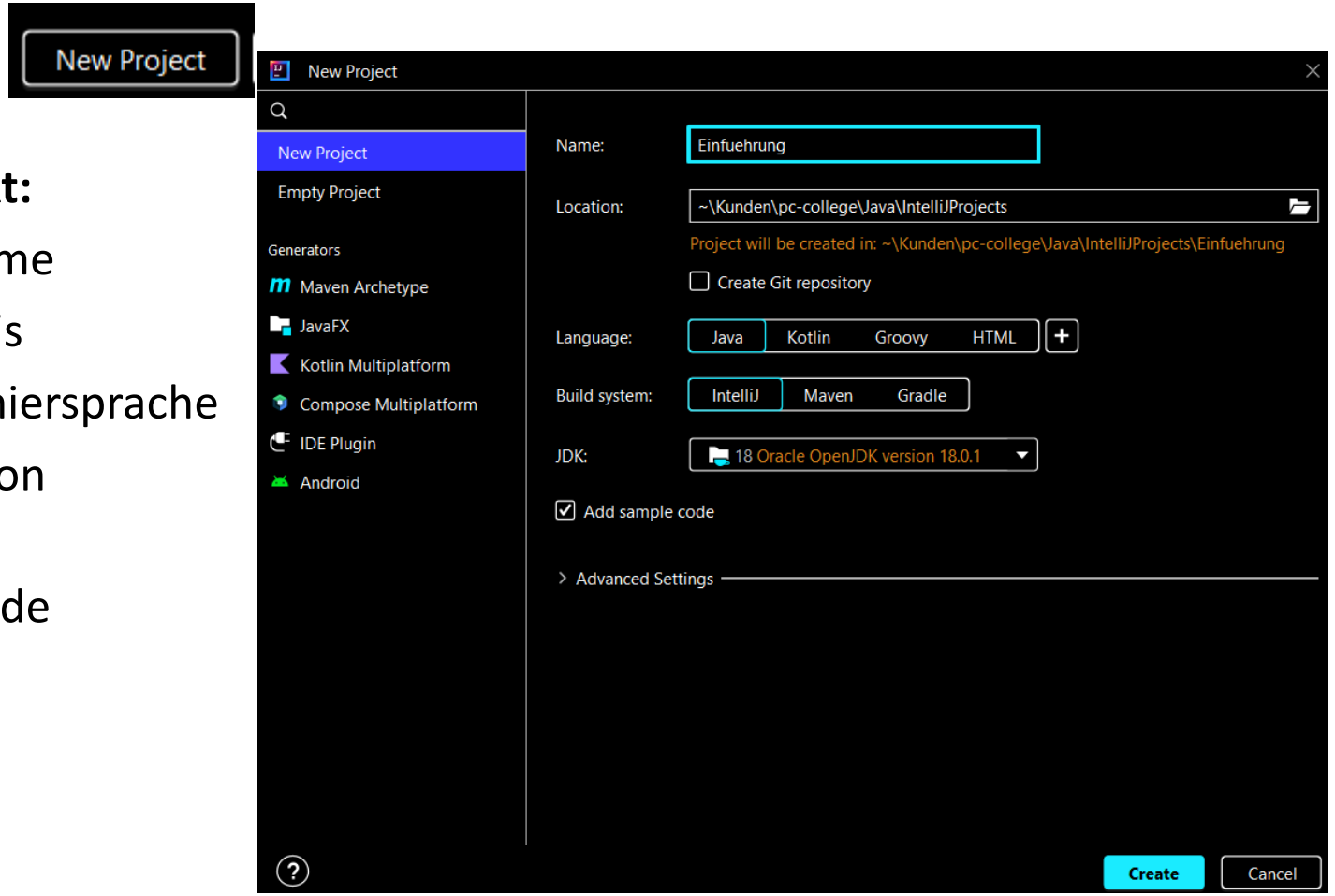
Start-Methode: main

Konventionen für Bezeichnungen

- Buchstaben a, …, z, A, …, Z; Umlaute erlaubt, nicht empfohlen
- Sonderzeichen: Unterstrich, Dollar, Zahlen 0, …, 9
- 1. Zeichen: Buchstabe, Unterstrich oder Dollar (keine Zahl)
- Kein Schlüsselwort
- Kamelnotation (camel case) sinnvoll

Bestandteil	1. Buchstabe	Besonderheit
Project	Groß	
Package	Klein	Umgedrehte URL + Name(n)
Class	Groß	
Variable	Klein	
Methode	Klein	

Java-Projekt in IntelliJ Idea

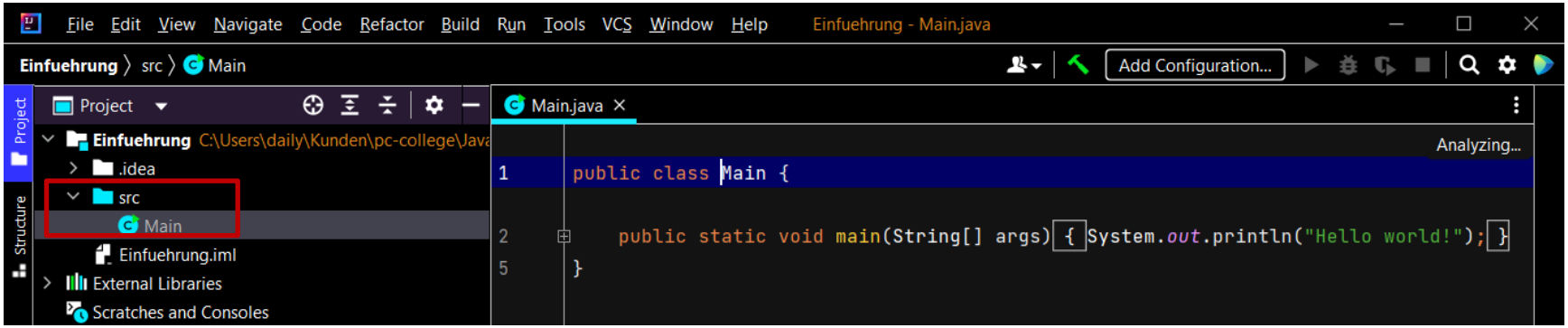


Neues Projekt:

- Projektname
- Verzeichnis
- Programmiersprache
- Java-Version
- Optional: Beispielcode

Erstelltes Projekt

Menü mit Registerkarten



Projekt-Explorer

u. a. src-Verzeichnis

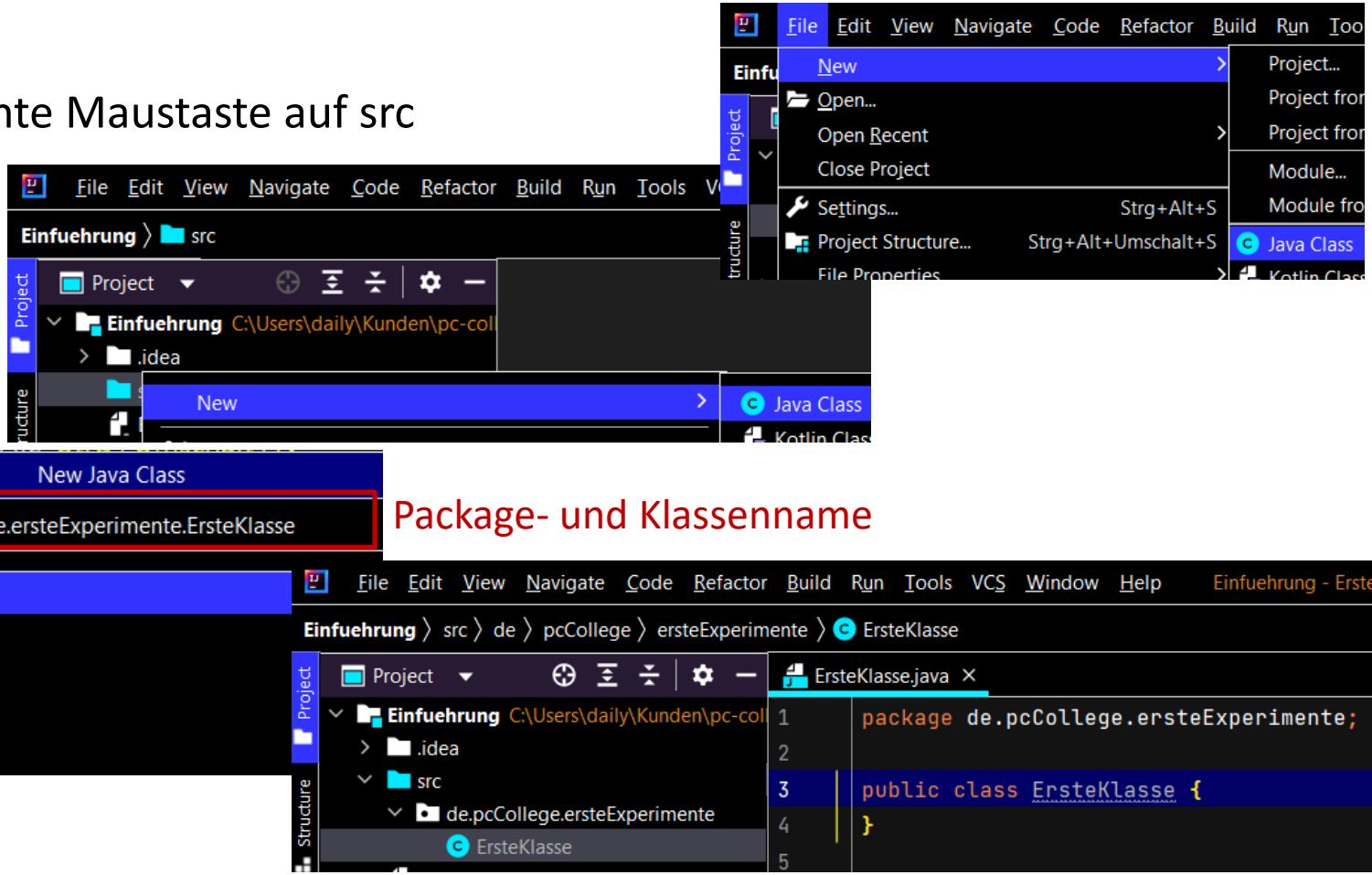
mit Java-Klassen

Code-Editor

Anlegen von Package und Klasse

Über File

oder rechte Maustaste auf src



Java-Schlüsselwörter

abstract	continue	for	new	switch
assert	default	~~goto~~	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
~~const~~	float	native	super	while

docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

Java-Klasse

```
public class KlassenName {
   // main-Methode mindestens 1x pro Projekt
   public static void main(String [] args) {
       System.out.println("Erste Ausgabe");
        /* … */
   }
   // Eventuell weitere Methoden
}
```

Ausgabe auf Konsole

Kommentare

// für einzeilige Kommentare

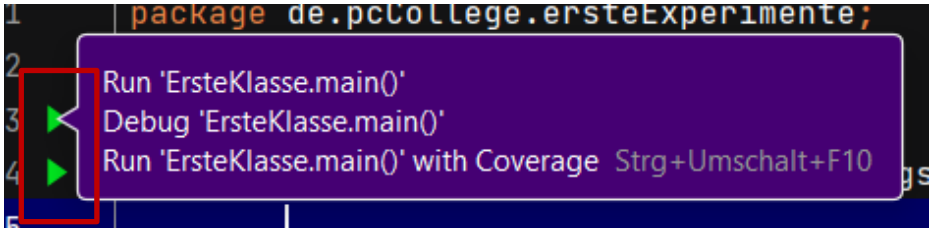
/* … */ für mehrzeilige I

/** … */ für JavaDoc

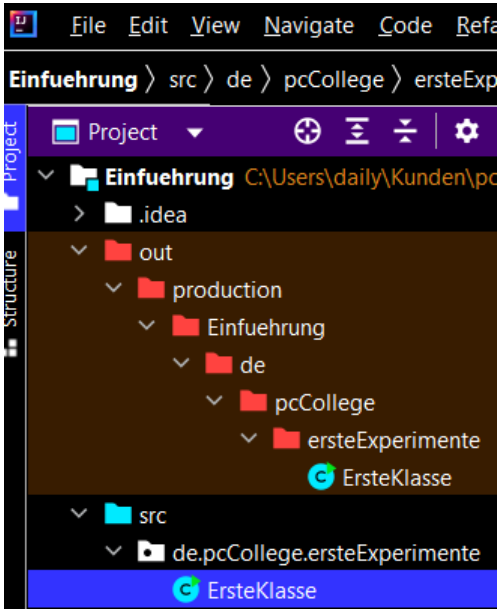
```
/**
 * @param args
 */
```

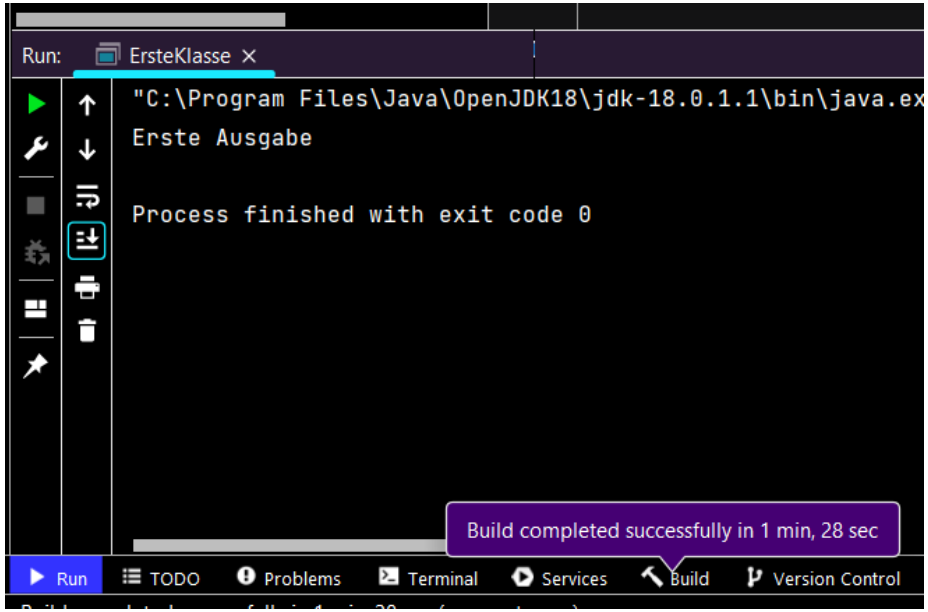
Autovervollständigung

Test



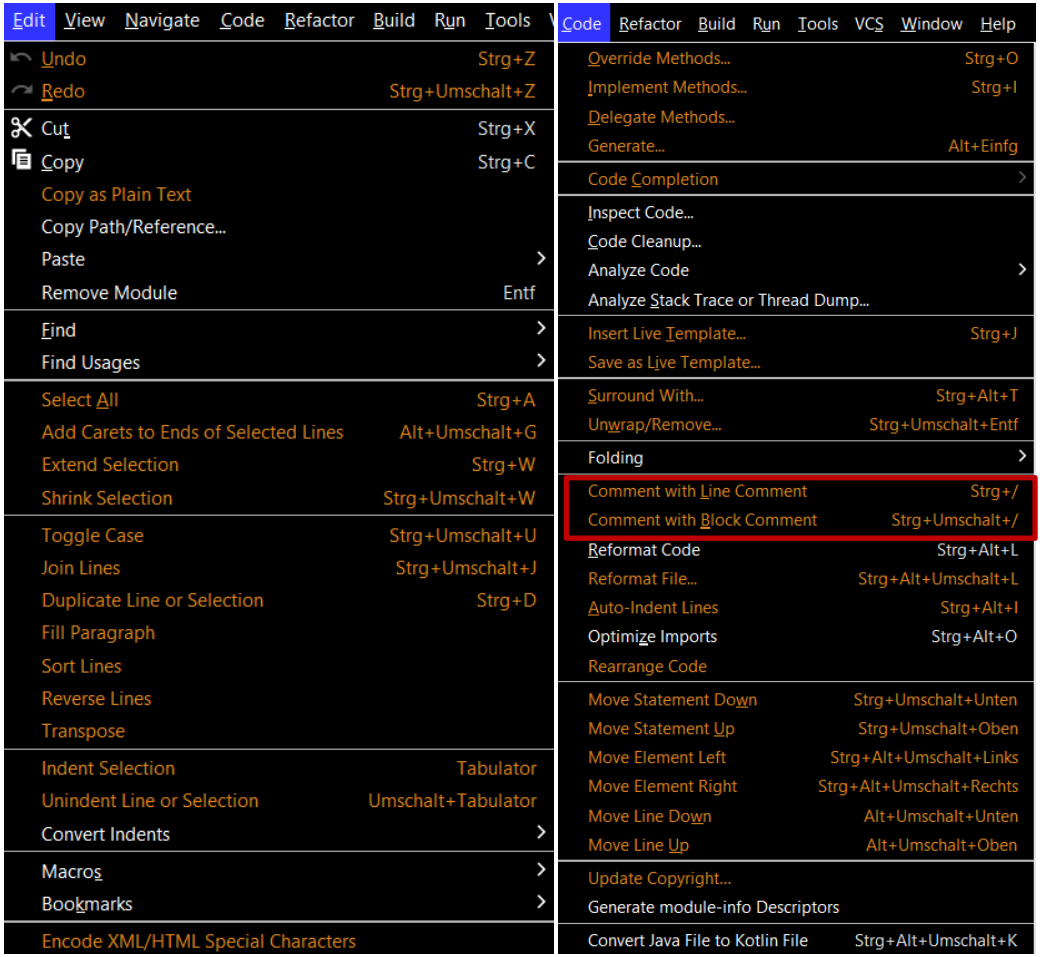
Kompilierter Bytecode





Beispiele für Shortcuts

resources.jetbrains.com/storage/products/intellij-idea/docs/IntelliJIDEA_ReferenceCard.pdf



Erweiterung des Beispiels

```
// import aus anderem Package (vgl. Java API)
import java.util.Scanner;
public static void main(String[] args) {
        System.out.println("Bitte geben Sie Ihren
   Namen ein");
        Scanner eingabe = new Scanner(System.in);
        System.out.printf("Guten Morgen, Herr/Frau
   %s", eingabe.next());
        eingabe.close();
   }
```

Eingabevorbereitung

Formatierte Ausgabe

Eingabe

Bewertung

Programmablauf: **E**ingabe – **V**erarbeitung – **A**usgabe

Aktuelles Beispiel:

- Einmalige Eingabe
- Kann nicht weiter verarbeitet werden

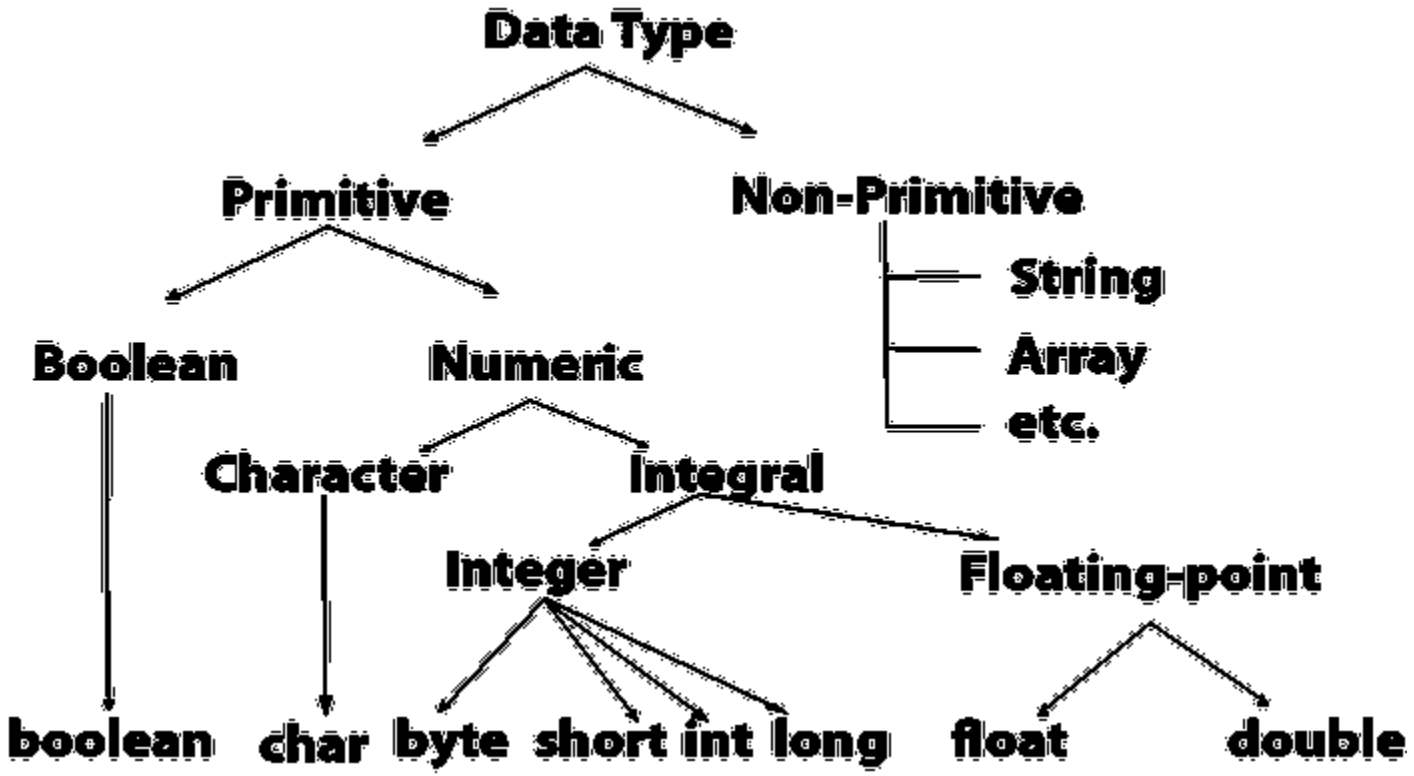
→ Verwendung von **Variablen**

- "Behälter", können unterschiedliche Werte aufnehmen
- Ermöglichen werteabhängige Weiterverarbeitung
- In Java: Haben einen **Datentyp**

→ Programme nicht nur für einen Wertesatz
Unterschiedliche Ausgaben in Abhängigkeit von Eingaben

3.2 Datentypen

www.javatpoint.com/java-data-types



(Primitive) Datentypen

docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Typname	Größe	Wrapper-Klasse	Wertebereich
boolean	undefiniert	java.lang.Boolean	true / false
char	16 bit	java.lang.Character	0 ... 65.535 (z. B. 'A')
byte	8 bit	java.lang.Byte	-128 ... 127
short	16 bit	java.lang.Short	-32.768 ... 32.767
int	32 bit	java.lang.Integer	-2.147.483.648 ... 2.147.483.647
long	64 bit	java.lang.Long	-2^{63} bis $2^{63}-1$, ab Java 8 auch 0 bis $2^{64}-1$
float	32 bit	java.lang.Float	+/-1,4E-45 ... +/-3,4E+38
double	64 bit	java.lang.Double	+/-4,9E-324 ... +/-1,7E+308

Beispiele für wichtige Klassen
(nicht primitive)

- Wrapper-Klassen: z. B. für Typumwandlungen
- Scanner für Eingabe
- String: Zeichenkette
- Klassen für Datum/Zeit
- GUI-Klassen
- Zufallszahl: Random

Variable und Operatoren

- Variable, z. B.
```
int zahl;
boolean stopp;
```
- Feld(Array), z. B.
```
float [] feld1 = {value1, … };
float [] feld2 = new float [5];
feld2[2] = 3.5f; feld2[3]=7.8f;
```
- Zuweisung, z. B.
```
zahl = 5;
stopp = false;
feld2[0] = 3.9f;
```
- Operatoren
  Arithmetisch: `+, -, *, /, %, ++, --, +=, -=, …`
  Logisch: `!, &, &&, ^, |, ||`

Typumwandlungen: Beispiele

```
String z1 = "50";
int iZahl = Integer.parseInt(z1);
String z2 = "3.5";
float fZ = Float.parseFloat(z2);
double dZ = Double.parseDouble(z2);
System.out.println("Ganze Zahl " + iZahl);
System.out.println("Fließkommazahlen " + fZ + " " +
  dZ);
```

Beispiele


```
// Addition in einer neuen Klasse "Rechner"
Scanner eingabe = new Scanner(System.in);
System.out.println("Bitte geben Sie die erste Zahl ein");
int z1 = eingabe.nextInt();
System.out.println("Bitte geben Sie die zweite Zahl ein");
int z2 = eingabe.nextInt();
System.out.printf("%d + %d = %d", z1, z2, z1+z2);
eingabe.close();
// Arrays
String [] wochentage = {"Montag", "Dienstag", "Mittwoch",
   "Donnerstag", "Freitag", "Samstag", "Sonntag"};
System.out.println(wochentage[0]);
int [] alter = {25, 33, 47, 28, 36};
```

Aufgabenblatt 1

Siehe pdf-Datei

3.3 Bedingungen

Basis: Aussagenlogik

- (Gegen-)Beispiele für Aussagen:
  „Im Moment scheint die Sonne.“
  ~~„Sonnenstrahlen wärmen wunderbar.“~~
- Objektive Aussage ist „**wahr**“ oder „**falsch**“
  (keine „Grautöne“, keine Gefühle)
- **Logische Operatoren**:
  !, &, &&, ^, |, ||

Zeichen	Bedeutung
!	Nicht
&	Bitweises Und
&&	Logisches Und
^	Bitweises XOR
|	Logisches XOR
||	Logisches Oder

Einsatzgebiete:

- Verzweigungen
- Schleifen

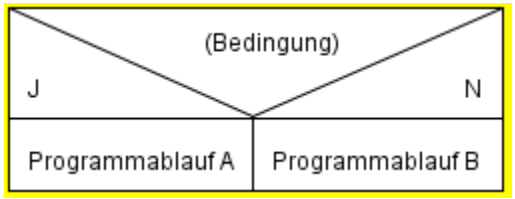
} Kontrollstrukturen

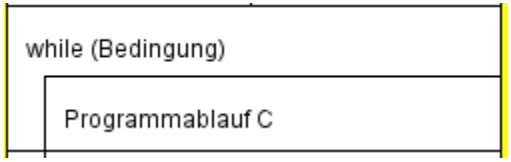
Kontrollstrukturen

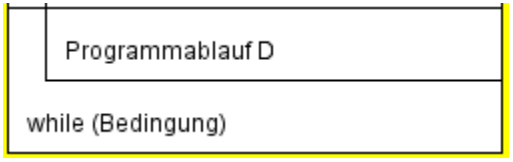
Zweck

- Unterschiedliche Programmabläufe (Verzweigung)
- Wiederholung von Programmabläufen (Schleife)

Modellierung: Nassi-Shneiderman-Diagramme







Kontrollstrukturen in Java

- Verzweigung :

```
if(<bedingung>){ … } else { … }
switch (<variable>){
    case nn: …; break; … }
```

Ternär: `(<bedingung>)? …:…;`

Beispiel: `int minVal = (a < b) ? a : b;`

- Schleifen :

Kopfgesteuert:

```
for(int i=0;i<arrayName.length; i++) {    … }
```

Schleifenindex i; Schrittweite: Standard 1, andere z. B.: `i+=2`

```
while (<bedingung>){ … }
```

Fußgesteuert:

```
do { … } while (<bedingung>);
```

Beispiel zu if … else

```
/* Zu einer vorgegebenen Zahl
 * soll ausgegeben werden,
 * ob diese kleiner als 18 ist.
 */
Scanner in = new Scanner(System.in);
int zahl = Integer.parseInt(in.next());
if(zahl < 18) {
      System.out.println("Zahl ist kleiner als 18");
}
else {
      System.out.println("Zahl ist gleich
            oder größer als 18");
}
```

Beispiel zu switch

```
/* Zu einer zufällig bestimmten Zahl zwischen 1 und 7
 *  soll der Name des entsprechenden Wochentags
 *  ausgegeben werden.
 */
Random zufall = new Random();
int tag = zufall.nextInt(6)+1;
switch (tag) {
      case 1: System.out.println("Montag"); break;
      case 2: System.out.println("Dienstag"); break;
      …
      case 7: System.out.println("Sonntag"); break;
      default:System.out.println("kein Tag"); break;
}
```

Zwischenfrage

Wie lässt sich die switch-Anweisung mit Hilfe eines Arrays ersetzen?

Beispiel zu for

```
/* Für die Zahlen zwischen 1 und 10
 *  sollen die Quadrate
 *  ausgegeben werden.
 */
for (int i = 1; i <= 10; i++) {
   System.out.println("Quadrat von "+i+" = "+(i*i));
}
```

Beispiel zu while

/* Der Durchschnitt eingegebener positiver Zahlen soll ausgegeben werden.
 * Bei Eingabe einer negativen Zahl stoppt das Verfahren.
 */

```
int sum = 0; int zaehler = 0;
Scanner in = new Scanner(System.in);
int zahl = 0;
while ((zahl= Integer.parseInt(in.nextInt()) > 0) {
        sum += zahl;
        zaehler++;
}
if(zaehler > 0) {
        System.out.println("Durchschnitt " + (sum/zaehler));
} else {
        System.out.println("Keine Zahlen eingegeben");
}
```

Beispiel zu do … while

/* Der Durchschnitt eingegebener positiver Zahlen soll ausgegeben werden.
 * Bei Eingabe einer negativen Zahl stoppt das Verfahren.
 */

```
int sum = 0; int zaehler = 0;
Scanner in = new Scanner(System.in);
int zahl = 0;
do {
    zahl = Integer.parseInt(in.next());
    sum += zahl;
    zaehler++;
} while (zahl > 0);
if(zaehler > 1) {
        System.out.println("Durchschnitt " +
                ((sum-zahl)/(zaehler-1)));
} else {
        System.out.println("Keine Zahlen eingegeben");
}
```

Bewertung

Welche Gemeinsamkeiten und welche Unterschiede finden Sie?

Aufgabenblatt 2

Siehe pdf-Datei

3.4 Fehlerbehandlung

Fehlertypen:

- **Typografisch** „Tiffpeler“
  teilweise von IntelliJ Idea erkannt
- **Syntaktisch** ("falsche Grammatik") „Wir hat Hunger“
  von IntelliJ Idea erkannt
- **Semantisch/logisch** „Die Maus frisst die Katze“
  teilweise von IntelliJ Idea erkannt
- Fehler durch äußere Umstände (**Laufzeitfehler**) „In Corona-Zeit: ÖPNV ohne Maske“

Beispiele:

Deutsche Bezeichnung für Variable als Typo markiert

for-Schleife double gehalt = **5000;** gehalt <= **3500 ;** als Fehler markiert

Gegenmaßnahmen

- Planung, Versionierung
- **Entwicklungswerkzeuge** (Syntax Highlighting, Codevervollständigung)
- **Aussagekräftige Bezeichner, Formatierung, Kommentare, „KISS“** (keep it simple and stupid)
- Fehler abfangen und melden: **try-catch**, throw/s, **finally**
- Fehleranalyse**: Fehlersuche, Debugging**, **JUnit-Test**
- **Fehlerbehebung**

try … catch … finally

Zweck

Abfangen von möglicherweise auftretenden Fehlern,
z. B. bei Division durch 0, Problem bei Zugriff auf Datei/Datenbank/Netzwerk

Beispiel

```
double z1 = 5;
double z2 = 0;
try {
   System.out.println(z1/z2);
}
catch (ArithmeticException e) {
   System.out.println(e.getMessage());
}
finally {
   System.out.println("Das war der Versuch einer
       Division"); // wird immer ausgeführt
}
```

Zwischenaufgabe

Welche Fehlermeldung erhalten Sie beim Versuch mit einem falschen Index auf ein Array zuzugreifen?

Debugging

Zweck

- Nachverfolgen von Programmabläufen
- Überprüfen von Variableninhalten

Vorgehen

- Setzen von Haltepunkten (breakpoints)
- Evtl. Bedingungen an Haltepunkten definieren

Breakpoint

www.jetbrains.com/help/idea/using-breakpoints.html

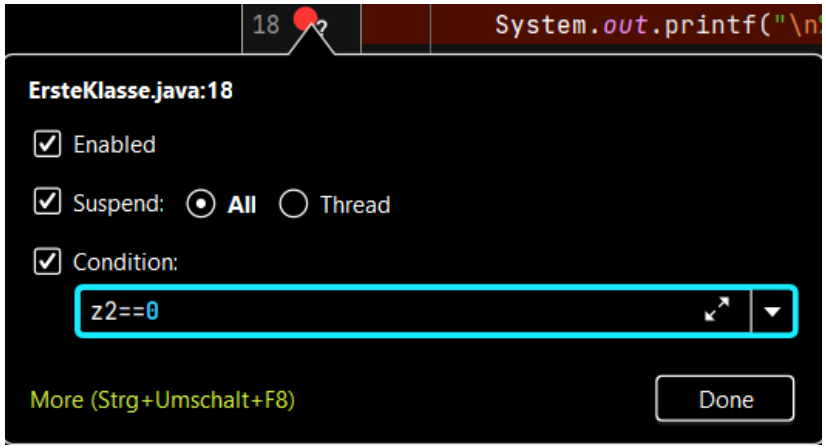
Breakpoint-Typen:

- Zeilen-Breakpoint
- Methoden-Breakpoint
- Beobachtungspunkt für Feld
- Breakpoint für Exception

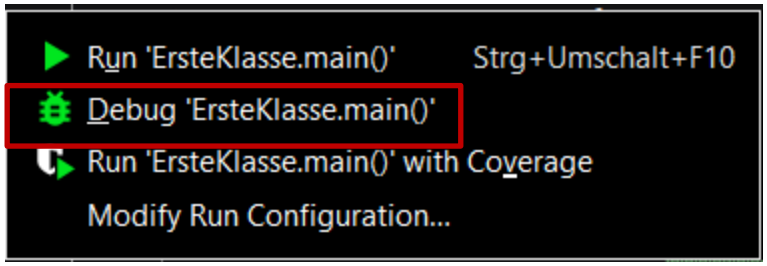
Setzen eines Breakpoints: Strg F8

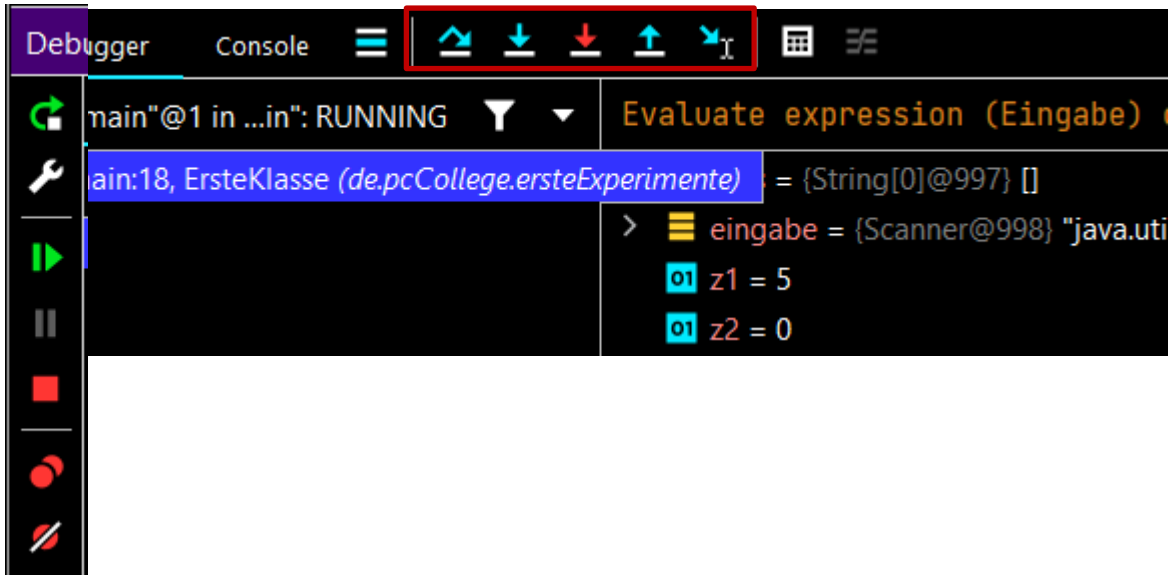
Zusätzlich Bedingung möglich

```
11      System.out.println("Bitte geben Sie die erste Zahl ein");
12      int z1 = Integer.parseInt(eingabe.next());
13      System.out.println("Bitte geben Sie die zweite Zahl ein");
14      int z2 = Integer.parseInt(eingabe.next());;
15      System.out.printf("\n%d + %d = %d", z1, z2, z1+z2);
16      System.out.printf("\n%d - %d = %d", z1, z2, z1-z2);
17      System.out.printf("\n%d * %d = %d", z1, z2, z1*z2);
18      System.out.printf("\n%d / %d = %d", z1, z2, z1/z2);
19      System.out.printf("\n%d modulo %d = %d", z1, z2, z1%z2);
20      eingabe.close();
```



Debugging


Run 'ErsteKlasse.main()' Strg+Umschalt+F10
Debug 'ErsteKlasse.main()'
Run 'ErsteKlasse.main()' with Coverage
Modify Run Configuration...


Debugger
Console
main"@1 in ...in": RUNNING
Evaluate expression (Eingabe)
ain:18, ErsteKlasse (de.pcCollege.ersteExperimente)
= {String[0]@997} []
eingabe = {Scanner@998} "java.uti
z1 = 5
z2 = 0

Zwischenaufgabe

Überlegen Sie sich für einige der zuvor erstellten Programme sinnvolle Haltepunkte und testen im Debug-Modus.