PROGRAMMIEREN IN JAVA

fritz JOBST

6. Auflage

JAVA 7

EXTRA: Mit kostenlosem E-Book

»Ungemein präzise und didaktisch klug aufgebaut...« Wolfgang Treß, Online-Redakteur

Im Internet: Alle Beispielprogramme und alle Lösungen zu den Aufgaben

HANSER

Inhalt

Vorwort ..... **XI**

1 Der Einstieg in Java **1**

1.1 Erstellung und Ablauf von Programmen in Java 2
1.2 Das erste Java–Programm 3
1.3 Erstellung und Ablauf des ersten Programms 5
1.4 Ein erstes Applet 9
1.5 Zusammenfassung 12
1.6 Aufgaben 12

2 Elemente der Programmierung **13**

2.1 Daten erklären und verarbeiten 14
2.1.1 Schreibweisen für Deklarationen und Wertzuweisungen 17
2.1.2 Beispiel: Elementare Ausdrücke 18
2.1.3 Beispiel: Bereichsüberschreitungen 21
2.1.4 Typumwandlungen 22
2.1.5 Deklarationen mit dem `static`-Modifizierer 23
2.1.6 Namen und ihre Gültigkeit 24
2.2 Kontrollfluss 25
2.2.1 Verzweigung 25
2.2.2 Mehrfachverzweigung 30
2.2.3 Schleifen mit Vorabprüfung 32
2.2.4 Schleife mit Prüfung am Ende 36
2.2.5 Verlassen von Schleifen 38
2.2.6 Programmausnahmen 39
2.3 Methoden 43
2.3.1 Definitionen 43
2.3.2 Beispiele zum Einsatz von Methoden 46
2.3.3 Rekursion 49
2.4 Felder 54
2.4.1 Eindimensionale Felder 55

2.4.2 Suche in Feldern ... 60
2.4.3 Sortieren ... 63
2.4.4 Mehrdimensionale Felder ... 66
2.5 Operatoren in Java ... 69
2.5.1 Arithmetische Operatoren ... 70
2.5.2 Vergleiche und logische Verknüpfungen ... 70
2.5.3 Reihenfolge der Auswertung bei logischen Ausdrücken ... 70
2.5.4 Inkrement und Dekrement ... 71
2.5.5 Bitverarbeitung ... 71
2.5.6 Bedingter Ausdruck ... 72
2.6 ANSI-Escape-Sequenzen ... 73
2.7 Aufgaben ... 74

3 Objektorientierte Programmierung ... 75

3.1 Auf dem Weg zu Klassen ... 75
3.1.1 Wege zur Objektorientierung ... 75
3.1.2 Beziehungen zwischen Klassen ... 78
3.1.3 Oberklassen und Unterklassen ... 79
3.1.4 Klassen und Objekte ... 80
3.1.5 Abstrakte Klassen ... 81
3.1.6 Entwurf der Klassen ... 81
3.2 Klassen in Java ... 83
3.2.1 Eine Klasse zum Verwalten von Mitarbeitern ... 84
3.2.2 Konstruktoren ... 88
3.2.3 Wertzuweisung und Übergabe als Parameter ... 88
3.2.4 Statische Klassenelemente ... 90
3.2.5 Eingeschachtelte Klassen, innere Klassen ... 93
3.2.6 Autoboxing und Unboxing ... 94
3.3 Unterklassen und Polymorphie in Java ... 97
3.3.1 Definition von Unterklassen in Java ... 97
3.3.2 Lebenszyklus von Objekten ... 100
3.3.3 Wie funktioniert die Polymorphie? ... 102
3.3.4 Wertzuweisung und Cast-Anweisung ... 104
3.3.5 Klassen und Ausnahmen ... 105
3.3.6 Abstrakte Klassen: Design für Vererbung ... 108
3.4 Schnittstellenvererbung ... 112
3.4.1 Beispiel: Geometrische Objekte ... 114
3.4.2 Beispiel: Sortieren von vergleichbaren Objekten ... 115
3.4.3 Beispiel: Funktionen als Parameter ... 118
3.4.4 Beispiel: Kopieren von Objekten ... 120
3.5 Dynamische Erzeugung von Objekten ... 122
3.6 Generische Klassen in Java ... 127
3.7 Aufzählung von Konstanten mit enum-Klassen ... 130
3.8 Allgemeine Eigenschaften von und für Klassen in Java ... 133
3.8.1 Der `final`-Modifizierer ... 133
3.8.2 Die `finalize`-Methode ... 133

3.8.3 Packages mit `package`, `import` ... 133
3.8.4 Sichtbarkeit von Namen in Java ... 135
3.8.5 Wiederherstellung des Zustands eines Objekts: Serialisierung ... 135
3.8.6 Zusicherungen ... 137
3.9 Aufgaben ... 139

4 Grundlegende Klassen ... 141
4.1 Nützliche Klassen und Packages ... 141
4.1.1 Übersicht der Aufgaben des Packages java.lang ... 142
4.1.2 Zeichenketten in Java: String ... 144
4.1.3 Die Klasse System ... 147
4.1.4 Die Klasse Math ... 147
4.1.5 Formatierte Ausgabe in Java ... 148
4.1.6 Reflexion von Java-Programmen ... 149
4.1.7 Annotationen ... 151
4.1.8 Reguläre Ausdrücke ... 153
4.1.9 Protokollierung von Programmläufen: Logging ... 156
4.2 Verwalten von Objekten mit Sammlungen ... 158
4.2.1 Prinzip für die Aufbewahrung von Objekten ... 159
4.2.2 Sequentieller Zugriff: List, Set und SortedSet ... 165
4.2.3 Assoziativer Zugriff: Map ... 176
4.2.4 Nützliche Klassen und Methoden für Sammlungen ... 181
4.3 Anwendungsfälle ... 186
4.3.1 Sortierte Ausgabe aller Einträge in einem Verzeichnis ... 186
4.3.2 Projektarbeit: Querverweisliste in Java ... 187
4.3.3 Binäre Bäume ... 190
4.4 Aufgaben ... 193

5 Ein-/Ausgabe in Java ... 193
5.1 Prinzip der Ein-/Ausgabe in Java ... 196
5.1.1 Eingabe in Java ... 198
5.1.2 Ausgabe in Java ... 202
5.2 Fallstudien zu Ein-/Ausgabe ... 208
5.2.1 Bearbeiten von Textdateien ... 208
5.2.2 Daten im plattformneutralen Format verarbeiten ... 210
5.2.3 Durchlaufen aller Dateien in einem Verzeichnis ... 211
5.2.4 Zugriff auf die Einträge in einem ZIP-Archiv ... 213
5.3 Aufgaben ... 214

6 Nebenläufigkeit in Java: Threads ... 215
6.1 Einstieg in Threads in Java ... 215
6.2 Grundlagen zu Threads ... 217
6.2.1 Nutzen von Threads ... 218
6.2.2 Wettrennen ... 220
6.2.3 Zustände von Threads ... 224

6.2.4 Wichtige Methoden für Threads ... 225
6.3 Monitore in Java ... 229
6.3.1 Grundlagen des Monitor-Konzepts in Java ... 229
6.3.2 Anwendung der Monitore in Java ... 230
6.4 Anwendungsfälle ... 232
6.4.1 Erzeuger-Verbraucher-Kopplung ... 234
6.4.2 Leser-Schreiber-Problem ... 235
6.4.3 Semaphoren ... 237
6.5 Verklemmungen und die fünf Philosophen ... 239
6.6 Animationen ... 243
6.7 Aufgaben ... 247

7 Graphik-Anwendungen in Java ... 247
7.1 Struktur von GUI-Anwendungen ... 247
7.1.1 Ein erstes Programm für Swing ... 248
7.1.2 Prinzip der ereignisgesteuerten Programmierung ... 249
7.1.3 Hierarchie der Swing-Klassen für Steuerelemente ... 250
7.1.4 Elementare Steuerelemente ... 251
7.1.5 Das Model-View-Controller-Paradigma und Swing ... 252
7.1.6 Anordnung der Komponenten ... 253
7.2 Ereignissteuerung ... 254
7.2.1 Das „Delegation Event Model“ ... 256
7.2.2 Listener-Schnittstellen und Adapter ... 257
7.3 Layoutmanager ... 258
7.3.1 BorderLayout ... 259
7.3.2 FlowLayout ... 260
7.3.3 GridLayout ... 261
7.3.4 CardLayout ... 261
7.3.5 GridBagLayout ... 262
7.3.6 BoxLayout (nur Swing) ... 265
7.3.7 Schachtelung der Layouts ... 265
7.4 Steuerelemente in Benutzeroberflächen ... 266
7.4.1 Schaltflächen: JButton ... 267
7.4.2 Checkboxen und Radiobutton ... 268
7.4.3 Statischer Text zur Anzeige von Informationen ... 270
7.4.4 Listen zur Auswahl ... 270
7.4.5 Elementare Auswahl mit der Combobox ... 272
7.4.6 Textfelder ... 273
7.4.7 Menüs in Java ... 274
7.5 Steuerelemente unter der MVC-Architektur ... 276
7.5.1 Übersicht: Aufgabenverteilung Swing-Anwender ... 278
7.5.2 Vertiefung für JList und JComboBox ... 278
7.5.3 Tabellen und Baumsteuerelemente ... 283
7.6 Kurs: GUI-Anwendungen ... 293
7.6.1 Erstellung einer grafischen Komponente ... 294
7.6.2 auf Maus-Klicks ... 296

7.6.3 Reaktion auf Mausbewegungen: ein Malprogramm 298
7.6.4 Turtle-Graphik 300
7.6.5 Dialoge in Java 304
7.6.6 Datei-Dialoge in Java 308
7.6.7 Die Türme von Hanoi 310
7.7 Aufgaben 314

8 Programmierung in Netzwerken 321
8.1 Programmierung von Sockets 321
8.1.1 Grundlagen von Netzwerken 322
8.1.2 Sockets in Java 323
8.2 Java-Programme für Internet-Protokolle 330
8.2.1 Ein Client für http 330
8.2.2 Ein Client für ftp 334
8.2.3 Ein Client für smtp 335
8.3 RMI 337
8.3.1 Prinzip von RMI 338
8.3.2 Praktische Durchführung mit RMI: ein Addierservice 338
8.4 Aufgaben 341

9 Anbindung von Datenbanken mit JDBC 343
9.1 Grundlagen von JDBC 344
9.1.1 Grundsätzlicher Ablauf beim Zugriff 344
9.1.2 Einstieg in relationale Datenbanken und SQL 347
9.1.3 Klassen und Schnittstellen im Package java.sql 349
9.2 Beispiel: Die Datenbank mit den Speisen in JDBC 356
9.2.1 Programmierung der Verbindung zu den Datenbanken 356
9.2.2 Vorbereitung: Datenbanken einrichten 358
9.2.3 Zugriffe mit JDBC 359
9.3 Datentypen in Java und SQL 362
9.4 Metadaten 362
9.4.1 Metadaten und die Auskunft über die Datenbank 363
9.4.2 Anwendung 364
9.5 Aufgaben 366

10 Bearbeiten von XML in Java 369
10.1 Schreiben von XML und Lesen mittels JAXB 370
10.1.1 Zusammenhänge: Klasse und Objekt bzw. Schema und XML 371
10.1.2 Kochrezept: Anleitung zur Benützung von JAXB 372
10.2 SAX-Parser 374
10.3 Aufgaben 377

11 Werkzeuge für die Java-Programmierung 379
11.1 Der Compiler javac 379
11.2 Der Interpreter java 380

11.3 Applet-Viewer 380
11.4 Der Generator für die Dokumentation 381
11.5 Der Disassembler 381

Literatur 383

Register 385

1

1 Der Einstieg in Java

Dieses Buch will Ihnen helfen, einfache Probleme mit Programmen in Java zu lösen und diese Programme ablaufen zu lassen. Dazu führt dieses Kapitel von den prinzipiellen Möglichkeiten von Java bis zu einem ersten Beispiel einer Anwendung.

Welche Probleme kann man mit Java lösen?

Eines Tages fand ich an der Tür zur Küche eine Mitteilung meines Sohnes mit folgendem Inhalt:

```
if (hunger)
  {
  goto kühlschrank;
  open door;
  eat leberkas;
  }
```

Dieser Zettel besteht aus einzelnen elementaren Anweisungen, die Schritt für Schritt von einem Menschen auszuführen sind: falls Hunger vorhanden ist, gehe man zum Kühlschrank, öffne die Tür und entnehme und verzehre Lebensmittel.[1] Solche Folgen von Anweisungen nennt man auch Programme. Die Schreibweise von Programmen folgt anscheinend einer gewissen Syntax. Es sind also die Regeln einer Programmiersprache einzuhalten, die sich in obigem Fall an meiner Vorliebe für Java orientierte.

Die Anweisungen in einem Computer-Programm richten sich hingegen an die CPU[2] eines Rechners und müssen deswegen so elementar aufgebaut sein, dass eine einfache Maschine sie ausführen kann. Eine CPU arbeitet mit einem Speicher (sog. RAM[3]) mit wahlfreiem Zugriff auf einzelne Daten zusammen. Die CPU holt Daten vom Speicher, verarbeitet sie und legt die Ergebnisse wieder im Speicher ab. Die einzelnen CPU-Befehle dienen z. B. dem Laden von Daten, der Verarbeitung sowie dem Abspeichern der Ergebnisse. Auf

---

[1] Bei genauerer Betrachtung fehlt eine Anweisung zum Schließen der Kühlschranktür.

[2] CPU: Central Processing Unit, Zentrale Verarbeitungseinheit: sie führt die Anweisungen aus

[3] RAM: Random Access Memory, Speicher mit wahlfreiem Zugriff.

einem üblichen PC würde man zwei ganze Zahlen, die im Speicher an den Stellen a bzw. b liegen, wie folgt addieren und die Summe im Speicher an der Stelle c ablegen:

```
mov eax,a ; Transportiere Inhalt von a nach Register[4] eax.
add eax,b ; Addiere Inhalt von b auf Register eax.
mov c,eax ; Speichere eax im RAM bei c ab.
```

Diese Anweisungen für eine einfache Addition sind in der maschinennahen Programmierung ebenso umständlich wie fehleranfällig zu schreiben. Auf diese Weise könnten nur Computer-Artisten größere und lauffähige Programme erstellen. Zur Lösung dieses Problems entwickelte man die sog. höheren Programmiersprachen, bei denen man die Anweisungen in einer für den Menschen akzeptablen Form niederlegt. In einer höheren Programmiersprache wie Java würde man zwei Zahlen wie folgt addieren:

```
c = a + b; // Wertzuweisung: c ergibt sich zu a + b.
```

Diese für Menschen lesbare und verständliche Form eines Programms als Text ist für eine CPU unverständlich, man benötigt einen Übersetzer `javac`, einen sog. *Compiler*, der aus der Anweisung „c ergibt sich als Summe aus a und b“ die entsprechenden Maschinenbefehle erzeugt. Eine solche Übersetzung müsste man für jede CPU durchführen, auf der das Programm laufen soll. Deswegen entschieden sich die Java-Designer für die Konstruktion einer symbolischen universellen Maschinensprache,[5] bei der nach der Übersetzung das längste Stück des Weges schon zurückgelegt ist, die Befehle aber noch für die einzelnen Plattformen mit dem Interpreter `java` ausführbar sind: *Write once, run everywhere.*

1.1 Erstellung und Ablauf von Programmen in Java

Programme in einer Hochsprache bestehen aus einer Folge von Anweisungen in Form von Text, den man in Textdateien mit dem Zusatz `.java` als Sequenz der einzelnen Zeilen speichert. Eine solche Textdatei erstellt man nicht mit einem Textverarbeitungssystem wie *Writer* oder *Word*, sondern mit einem vergleichweise einfachen Werkzeug wie einem Texteditor. Unter Unix/Linux käme im Prinzip *vi*, unter Windows *notepad* in Betracht. Diesen Text übersetzt man dann mit dem Compiler `javac` in Maschinensprache. Die so erzeugten Befehle für die Maschine bringt man dann letztendlich zum Ablauf auf einem Rechner. Diese Arbeitsschritte lassen sich in einer IDE[6] zusammenfassen.

Beim Ablauf unterscheidet man zwischen eingebetteten und selbstständig lauffähigen Anwendungen. Die eingebetteten Anwendungen, sog. Applets, können in `.html`-Dokumente[7] ebenso wie Bilder eingefügt werden, sind aber im Gegensatz zu diesen aktiv. Ein Applet läuft auf dem Rechner des Anwenders ab, es nutzt die dort vorhandenen Ressourcen an Rechnerleistung aus.

---

[4] Register: eine CPU hat nur wenige dieser Speicherplätze, die so schnell wie die CPU sind.
[5] der sog. Bytecode, der in Dateien mit der Endung .class liegt
[6] IDE: integrierte Entwicklungsumgebung, Integrated Development Environment
[7] HTML: Hypertext Markup Language

Eine selbstständig lauffähige Anwendung in Java entspricht eher einem Programm wie z. B. einer Textverarbeitung, ist allerdings nur zusammen mit dem Interpreter `java` lauffähig. Dieser Interpreter muss die Anweisungen im Java-Bytecode auf dem Rechner des Anwenders Schritt für Schritt ausführen. Solche Anwendungen können ihrerseits im Textmodus im Konsolenfenster ablaufen. Sie können aber auch Grafik-Fenster aktivieren und dort mit Grafik arbeiten.

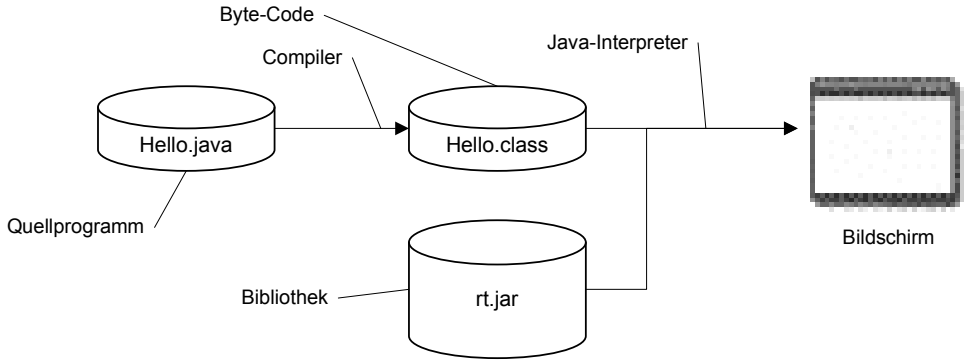

Abbildung 1.1 Erstellung und Ablauf eines Java-Programms

Der Compiler `javac` übersetzt in Hochsprache geschriebene Befehle der Textdatei `Hello.java` in die symbolische Maschinensprache (sog. Bytecode) für die diversen CPUs und legt sie in der Datei `Hello.class` ab. Zum Ablauf benötigt der Java-Interpreter `java` noch zahlreiche Hilfsprogramme, die in der Bibliothek `rt.jar` zusammengefasst zur Verfügung stehen. `rt.jar` gehört zum JDK[8] ebenso wie zum JRE[9]. Das JRE enthält alle zum Ablauf von Java-Programmen erforderlichen Bestandteile. Das JDK enthält darüber hinaus noch die Entwicklungswerkzeuge wie z. B. den Compiler `javac`. Die folgenden beiden Kommandos übersetzen das Programm Listing 1.1 und bringen es zum Ablauf.

```
javac Hello.java
java Hello
```

1.2 Das erste Java-Programm

Java-Programme sind Textdateien, die man mit einem Text-Editor wie *vi*, *emacs* oder *notepad* eingibt. Die Texte sind keineswegs wahlfrei, sondern folgen speziellen Regeln. Das erste Programm soll als Anwendung laufen und den Text „Hello World“ ausgeben. Hierzu muss das folgende Programm mit der Klasse `Hello` in der Datei `Hello.java` abgelegt werden. Der Dateiname muss aus dem Klassennamen mit dem Zusatz `".java"` bestehen.

[8] JDK: Java Devolopment Kit
[9] JRE: Java Runtime Environment

Er sollte nach einer Java-Konvention[10] mit einem Großbuchstaben beginnen. Leerräume sind im Namen in keinem Fall zulässig. Die Schreibweisen `Hello.java` für den Namen der Datei sowie `Hello` für den Namen der Klasse müssen eingehalten werden, `hello` ist ebenso falsch wie `HELLO` oder `hellO`, denn in Java muss man strikt auf die Unterschiede zwischen Groß- und Kleinschreibung achten.

Listing 1.1 Hello.java zum Einstieg in die Java-Programmierung

```
// Das erste Programm in der Datei Hello.java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Ein Java-Programm besteht aus einzelnen Wörtern der Sprache Java. **`public`** und **`class`** sind in Java reservierte Namen, `Hello` ist der Eigenname der definierten Klasse. **`public`** kennzeichnet einen „öffentlichen" Inhalt, den auch andere Java-Programme benützen dürfen. Mit **`class`** `Hello` eröffnet man eine neue Java-Klasse, die innerhalb der geschweiften Klammern `{}` folgt. In Kapitel 2 sind die reservierten Namen der Sprache Java zusammengestellt. Alle reservierten Namen von Java sind in diesem Buch fett gedruckt.

Ein Wort ist in Java eine Folge von Buchstaben bzw. Ziffern, die mit einem Buchstaben beginnt. Diese Worte werden nacheinander geschrieben. Worte können aus Folgen von Buchstaben und Ziffern bestehen, wobei das erste Zeichen ein Buchstabe sein muss. Es muss stets klar sein, wann ein Wort endet und das nächste folgt. Worte können auch nicht einfach getrennt werden. Die folgenden Zeilen sind deswegen fehlerhaft:

```
Publicstaticvoid
pub lic sta tic vo id
publicsta ticvoid
p u b l i c s t a t i c v o i d
```

In Java gibt es auch Wörter, die aus nur einem Zeichen bestehen, wie etwa die Zeichen `[ ] ( ) { } , . ;` Wenn das Ende eines Wortes eindeutig definiert ist, sind keine besonderen Trennzeichen zwischen aufeinanderfolgenden Worten, wie etwa ein Leerraum, erforderlich. Die folgenden unterschiedlichen Zeilen definieren die gleiche Funktionalität.

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

Die Schreibweise ist formatfrei. Zwischen zwei Worten können beliebig viele Leerzeichen, neue Zeilen oder Kommentare stehen. Die folgenden Zeilen sind korrekt formuliert, aber nicht gut lesbar, und nur zur Abschreckung angegeben:

```
public
static
void
main
(
String
```

[10] Siehe http://java.sun.com/docs/codeconv/index.html

```
args
[
]
)
{
```

Die Klasse `Hello` enthält eine Methode namens `main`. Sie können sich eine Methode[11] als eine in die Zeichen `{}` eingeschlossene Folge von Anweisungen vorstellen. Diese Folge besteht hier im Beispiel aus genau einer Anweisung. `main` ist das Hauptprogramm. Das Laufzeitsystem erwartet das Hauptprogramm `main` und aktiviert es bei Ausführung des Programms: die Anweisungen in `main` werden der Reihe nach durchlaufen. Im Beispiel wird eine Methode `System.out.println` zur Ausgabe eines Textes auf dem Bildschirm aufgerufen. Die Methode `println` befindet sich in der Bibliothek `rt.jar`.

Ein für sich alleine ablauffähiges Programm muss die `main`-Methode in obiger Form enthalten. `main` hat Zugriff auf die sog. Parameter `args` der Kommandozeile[12], macht aber in diesem Beispiel keinen Gebrauch davon.

Kommentare können in einer Zeile ab `//` eingefügt werden. Der Inhalt von Kommentaren unterliegt keiner Einschränkung. Wenn man Kommentare über mehrere Zeilen wünscht, schließt man den Text des Kommentars zwischen `/*` und `*/` ein.

1.3 Erstellung und Ablauf des ersten Programms

Zur Programmierung benötigt man das JDK, das man sich über die Adresse `java.sun.com` herunterladen kann und danach installieren muss. Mit dem JDK stehen sowohl der Interpreter als auch der Compiler zur Verfügung, müssten aber „von Hand“ benützt werden, wie in Kap. 11 beschrieben. Beachten Sie bitte, dass die JRE nicht zur Erstellung von Programmen ausreicht, da die Entwicklungswerkzeuge wie der Compiler fehlen. Eine IDE[13] bietet einen einheitlichen Zugriff all diese Werkzeuge. Der Aufwand bis zur Installation einer IDE scheint zunächst übertrieben, amortisiert sich aber bei der Arbeit mit Java in kürzester Zeit. In diesem Buch arbeiten wir mit Eclipse, das man unter `www.eclipse.org` findet. Die Eclipse-IDE gibt in diversen Varianten, für die Entwicklung mit Java reicht das einfachste Paket wie „Eclipse IDE for Java Developers“. Man erhält Eclipse als `.zip`-Archiv, das man durch Entpacken in ein Verzeichnis auf dem lokalen Rechner wie `c:\Programme\eclipse` installiert. Starten Sie dann `eclipse.exe`.

Nach dem Start fragt Eclipse nach dem Ordner für den Arbeitsbereich zum Ablegen aller erstellten Programme. Sie können den Ordner bei den eigenen Daten oder in einem beliebigen Verzeichnis anlegen. Wenn man bei künftigen Starts von Eclipse Fragen nach der Lage des Arbeitsbereichs vermeiden will, kann man das Häkchen bei der *Use this as default…*-Option anklicken.

---

[11] Methoden heißen in manchen Programmiersprachen auch Funktionen bzw. Prozeduren.
[12] Siehe Kapitel 11.
[13] IDE Integrated Development Environment