Programmieren in Java
Einfach Java lernen

DIE LESEPROBE

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 der Installation der Entwicklungsumgebung Eclipse über die 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 Computerprogramm 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)])

[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.

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 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 Computerartisten 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 vergleichsweise 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 zum Ab-

[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

lauf auf einem Rechner. Diese Arbeitsschritte lassen sich in einer IDE[6] zusammenfassen.

Bis zur Version 9 unterstützte Oracle eingebettete Anwendungen. Diese sog. Applets konnten in `.html`-Dokumente[7] ebenso wie Bilder eingefügt werden, liefen aber stets auf dem Rechner des Anwenders ab.

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 Grafikfenster aktivieren und dort mit Grafik arbeiten.

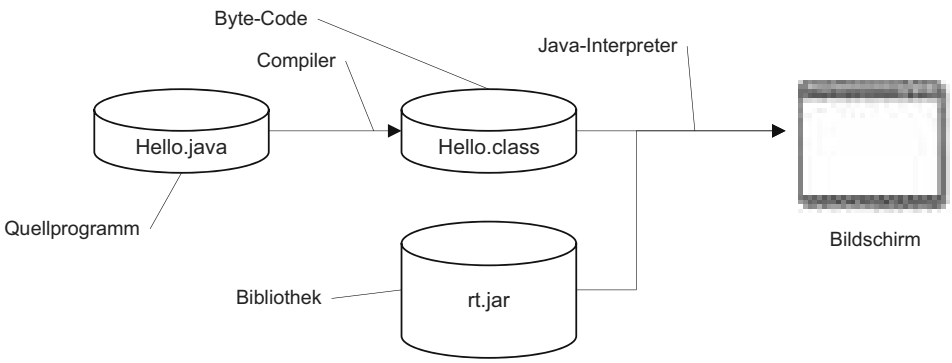

Bild 1.1 Erstellung und Ablauf eines Java-Programms

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 Bibliotheken wie z. B. `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`. Deswegen benötigen wir zum Programmieren das JDK und nicht nur das JRE. Wir greifen in diesem Buch nicht auf die o. a. Werkzeuge zurück, sondern benützen die Entwicklungsumgebung Eclipse. Die folgenden beiden Kommandos übersetzen das Programm Listing 1.1 und bringen es zum Ablauf.

---

6 IDE: integrierte Entwicklungsumgebung, Integrated Development Environment

7 HTML: Hypertext Markup Language

8 JDK: Java Development Kit

9 JRE: Java Runtime Environment

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

1.2 Das erste Java-Programm

Java-Programme sind Textdateien, die man mit einem Texteditor 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. 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
```

[10] Siehe *https://www.oracle.com/technetwork/java/codeconventions-150003.pdf*

In Java gibt es auch Wörter, die aus nur einem Zeichen bestehen, wie etwa die Zeichen [] () {} , . ; Wenn das Ende eines Worts 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 Wörtern 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
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 einer der Bibliotheken.

Ein für sich allein 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.

[11] Methoden heißen in manchen Programmiersprachen auch Funktionen bzw. Prozeduren.

[12] Siehe Abschnitte 2.2.3 und 2.4.1.

1.3 Erstellung und Ablauf des ersten Programms

Eine IDE[13] bietet einen einheitlichen Zugriff auf die Werkzeuge zur Programmerstellung. 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 der IDE Eclipse, die man unter *www.eclipse.org* findet. Die Eclipse-IDE gibt es in diversen Varianten, für die Entwicklung mit Java reicht das einfachste Paket wie „Eclipse IDE for Java Developers“. Diese IDE enthält bereits alle erforderlichen Komponenten, um Java-Programme zu entwickeln und auszuführen. Sie reicht für die Inhalte dieses Buchs aus. Unter Windows installiert man Eclipse mit dem Programm `eclipse-inst-jre-win64.exe`. Für die Eclipse-Version 2023-12 wird ein Verzeichnis auf dem lokalen Rechner im Bereich der Daten des angemeldeten Benutzers `C:\Users\--Benutzer--\eclipse\java-2023-12\eclipse` angelegt. Dort finden Sie das Programm `eclipse.exe`. Starten Sie dann `eclipse.exe`.

Nach dem Start fragt Eclipse nach dem Ordner für den Arbeitsbereich zum Ablegen aller erstellten Programme. 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.

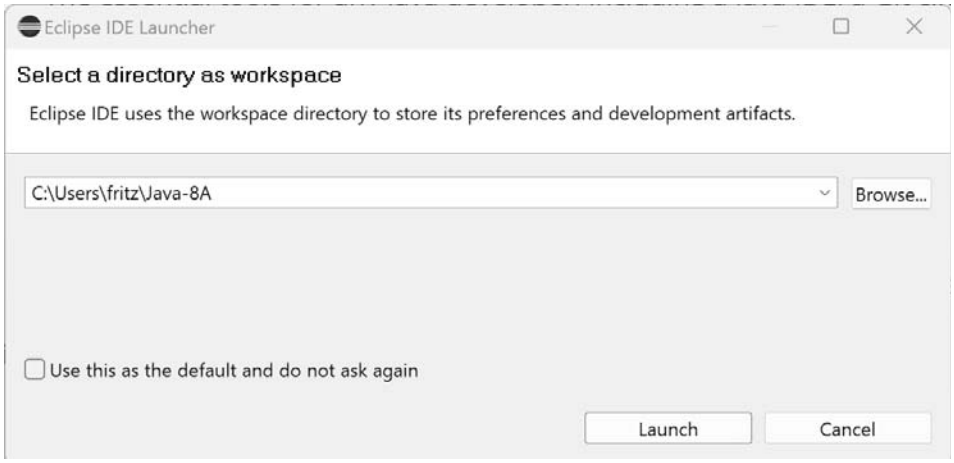

Bild 1.2 Auswahl eines Arbeitsbereichs

[13] IDE Integrated Development Environment

Danach meldet sich Eclipse mit seinem Willkommensbildschirm. Sie können jetzt das Eclipse-System erkunden oder gleich weiter zum Arbeitsbereich gehen, indem Sie auf das Pfeil-Symbol am rechten Rand des Bildschirms klicken. Danach sehen Sie die Arbeitsoberfläche von Eclipse aus Bild 1.3. Am linken Rand finden Sie den „Package Explorer“, mit dem Sie später die einzelnen Projekte erkunden können.

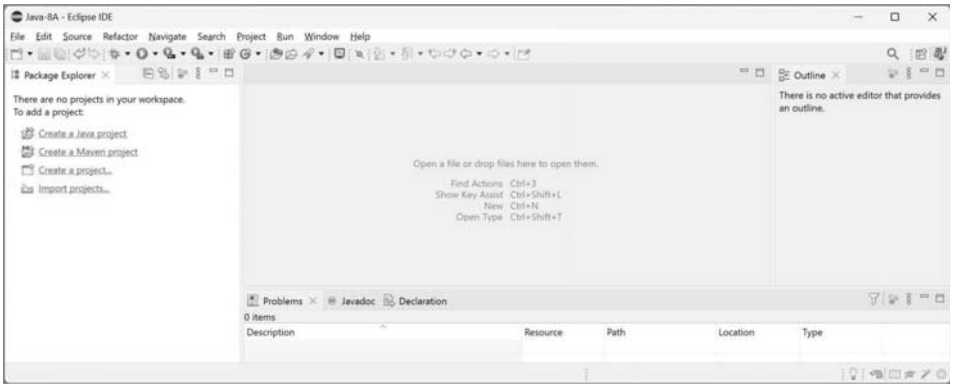

Bild 1.3 Arbeitsoberfläche von Eclipse

Eclipse organisiert die Arbeiten des Benutzers in sog. Projekten, sodass wir zunächst unter dem Menüpunkt *File/New/Java Project* ein neues Projekt erzeugen müssen. Ein Projekt kann mehrere Java-Programme enthalten. Diese Gliederung in Projekte erfordert einen zusätzlichen Schritt vor der Erstellung eines Java-Programms, ist aber auf die Dauer unerlässlich zum Ordnen der Programme und der dazugehörigen Dateien wie etwa Bilder.

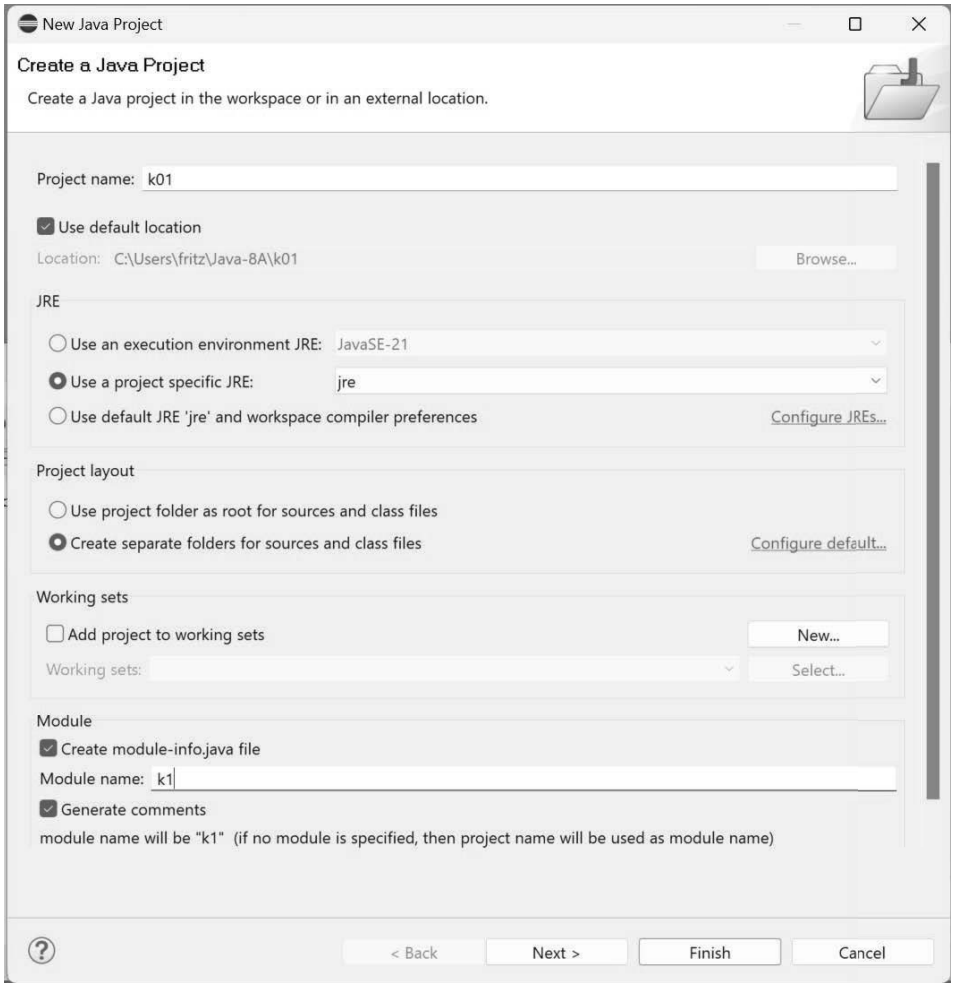

Bild 1.4 Anlegen eines Projekts in Eclipse

Unser erstes Projekt ist das Kapitel 1, für das wir das Projekt `k01` anlegen wollen. Es empfiehlt sich, einen eigenen Ordner für die vom Compiler erzeugten **`class`**-Files mit dem Binärcode anzulegen. Deswegen wählen wir unter *Project Layout* die Option *Create separate*. Damit vermeiden wir die Mischung aus Quellprogrammen und den Übersetzungsergebnissen. Achten Sie darauf, die Version JavaSE21 für Java 21 auszuwählen.

Danach kann man den Schalter *Finish* drücken und erhält die Ansicht aus Bild 1.5.

Da unser „Startprojekt“ leer ist, fügen wir durch einen Rechtsklick eine neue Klasse hinzu.

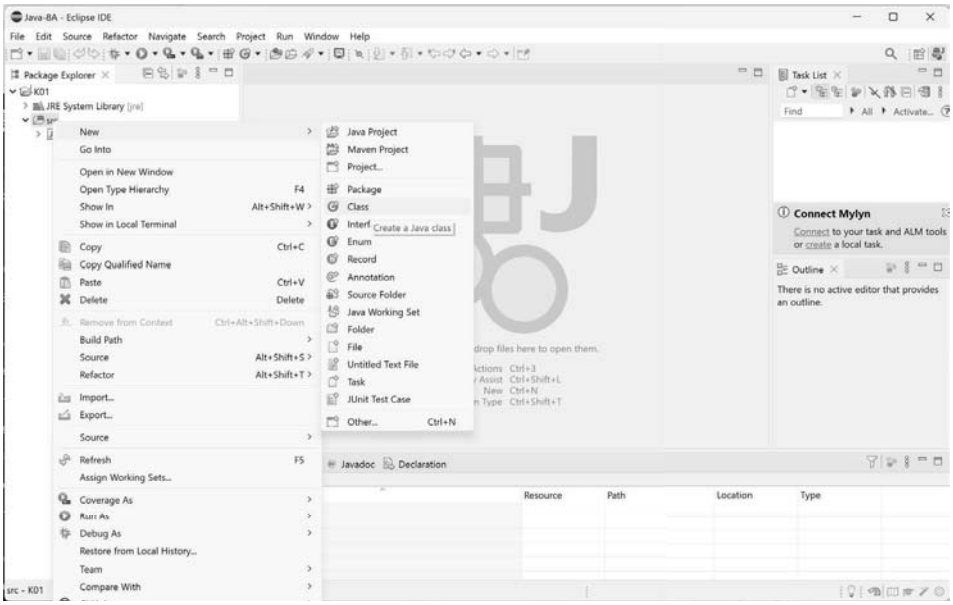

Bild 1.5 Anlegen einer Klasse in Eclipse

New/Class liefert den in Bild 1.6 angegebenen Dialog zur Erstellung einer Java-Klasse. Als Namen für unser erstes Package[14] im ersten Kapitel wählen wir wieder `k01`. Die Programme zu den einzelnen Kapiteln des Buchs finden Sie in den jeweiligen Projekten mit den entsprechenden Packages `kxy` unter der Nummer `xy` des Kapitels. Nach Drücken von *Finish* erhält man in Bild 1.6 den Rahmen für das erste Programm.

[14] Ein Package ist eine Zusammenfassung mehrerer Java-Programme in einem Unterverzeichnis.

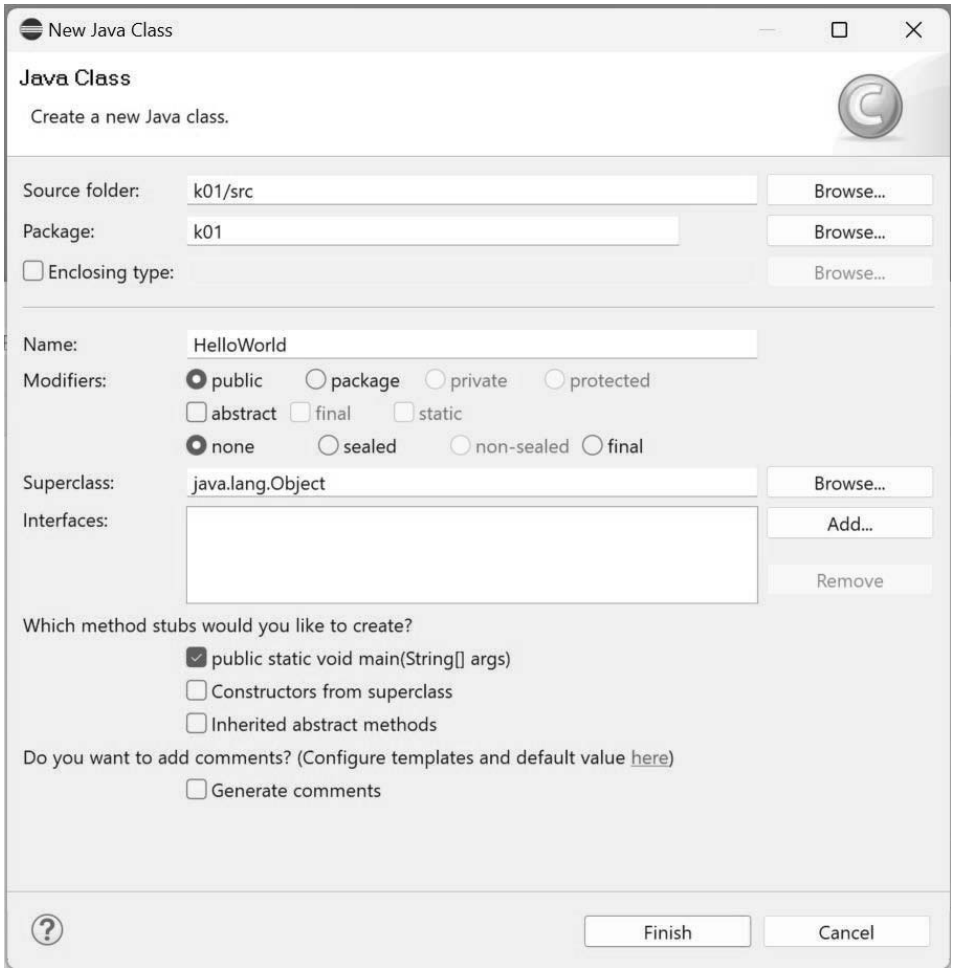

Bild 1.6 Dialog zum Erstellen einer Klasse

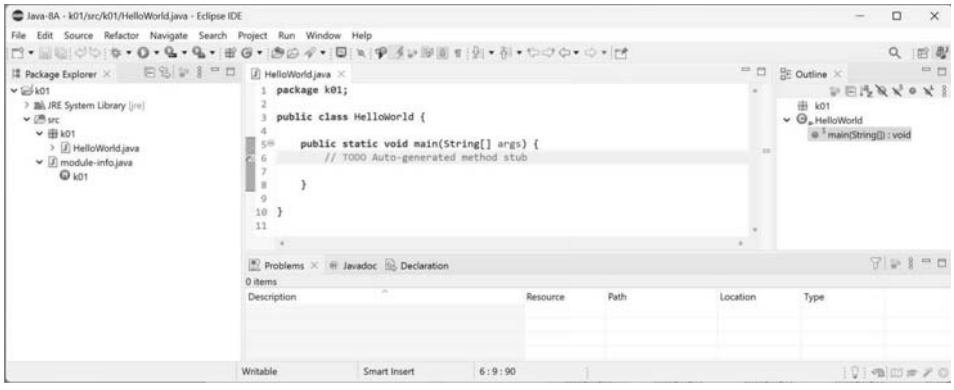

Bild 1.7 Eclipse liefert die erste Klasse im ersten Projekt

Jetzt können Sie die Anweisung `System.out.println ("Text");` in der Methode `main` in der Zeile 6 an der von Eclipse vorsorglich mit `TODO` markierten Stelle in Bild 1.7 eingeben. Der in runden Klammern angegebene Text ist ein sog. *Parameter* für den `println`-Befehl.

Eclipse übersetzt das Programm ohne besondere Aufforderung durch den Anwender. Nach einer Sicherung des Programms im Arbeitsbereich über das File-Menü oder mit der Tastenkombination **Strg+S** können Sie das Programm mit **Strg+F11** laufen lassen und erhalten im Konsolenfenster unten rechts in Bild 1.8 die Ausgabe des ersten Programms.

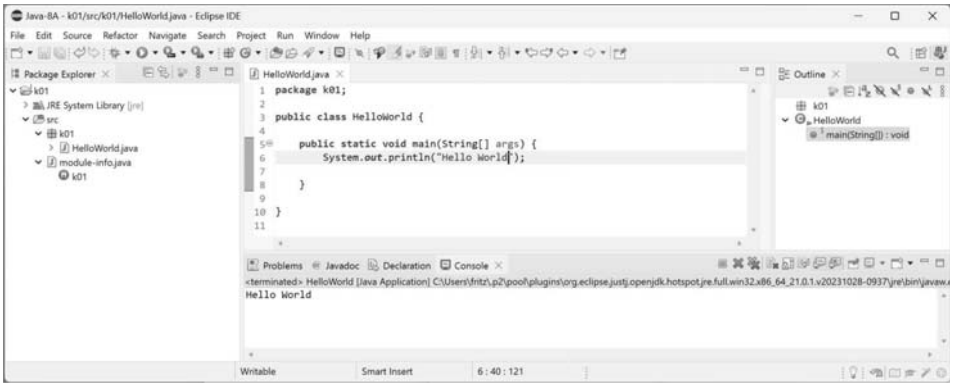

Bild 1.8 Das erste Programm mit Eclipse

1.4 Eine erste Anwendung mit Grafik

In Java ist der Schritt zu einer Anwendung für eine grafische Oberfläche kein Sprung mehr wie in anderen Systemen. Allerdings müssen wir die Hilfsmittel `JPanel` und `Graphics` importieren. Damit der Import erfolgreich ist, müssen wir in der in Bild 1.8 gezeigten Datei `module-info` das benötigte Modul `java.desktop` anfordern:

```
module k01 {
  requires java.desktop;
}
```

Das folgende Programm in Listing 1.2 soll in der Mitte des Bildschirms einen Text ausgeben. Der Text muss gezielt in der Mitte des Bildschirms platziert werden.

Statt in der `main(…)`-Methode in Listing 1.1 schreiben wir unsere Befehle in der `paint(Graphics g)`-Methode in Listing 1.2.

Listing 1.2 Die erste Anwendung in Java mit Grafik

```
package k01;
import javax.swing.JPanel;
import java.awt.Graphics;

// Eine erste Anwendung mit Grafik in Java
public class HelloWorldScreen extends JPanel {

  @Override
  public void paint(Graphics g) {
    // Hier müssen wir die Befehle der Form g.befehl(…); eintragen
    g.drawString("Hello World", getWidth() / 2, getHeight() / 2);
  }

  public static void main(String[] args) {
    MyPanelViewer.startGraphic(new HelloWorldScreen());
  }
}
```

Listing 1.3 Die Klasse MyPanelViewer

```
package k01;

import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;

//Ein JFrame zur Darstellung eines JPanel
public class MyPanelViewer extends JFrame {

  public MyPanelViewer(JPanel panel) {
    super("Start Grafiksystem");
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(300, 200);
    add(panel, BorderLayout.CENTER);
    setVisible(true);
  }

  public static void startGraphic(JPanel panel) {
    javax.swing.SwingUtilities.invokeLater(() -> new MyPanelViewer(panel));
  }
}
```

Problemanalyse

Das Programm `MyPanelViewer` in Listing 1.3 soll JPanels am Bildschirm anzeigen. Wir benötigen das Programm zur Anzeige von Grafik und erklären die Prinzipien in Kapitel 7.

`HelloWorldScreen` ist ein `JPanel`. Diese in Kapitel 3 eingeführte „Ist-ein-Beziehung“ formulieren wir so:

```
class HelloWorldScreen extends JPanel
```