Einführung in die Informatik

Jumping into Java

Programme, Modelle, Objekte, Klassen, Methoden

Cyrill Stachniss
Wolfram Burgard

Java, eine moderne, objektorientierte Sprache

Beispielprogramm:

```
class Program1 {
   public static void main(String[] arg) {
        System.out.println("This is my first Java program");
        System.out.println("but it won't be my last.");
   }
}
```

Ausgabe des Programms `Program1`:

```
This is my first Java program
but it won't be my last.
```

Modelle

Modelle sind (vereinfachte) Darstellungen. Sie enthalten die relevanten Eigenschaften eines modellierten Objektes.

Beispiel Kundendienst:

- Stadtplan.
- Nadeln markieren die Standorte der Kundendienstmitarbeiter.
- Fähnchen markieren die Stellen, an denen Kunden warten.

Abstraktes, unvollständiges Modell, welches aber für die Zuordnung von Mitarbeitern zu Kunden gut geeignet sein kann.

Eigenschaften von Modellen

- **Elemente eines Modells repräsentieren andere, komplexe Dinge.**
  Nadeln repräsentieren die Kundendienstmitarbeiter.
- **Die Elemente eines Modells haben ein bestimmtes, konsistentes Verhalten.**
  Nadeln repräsentieren Positionen und können an andere Stellen bewegt werden.
- **Die Elemente eines Modells können entsprechend ihrem Verhalten zu verschiedenen Gruppen zusammengefasst werden.**
  Mitarbeiter fahren von Kunde zu Kunde, Kunden kommen hinzu und verschwinden wieder.
- **Aktionen von außen stoßen das Verhalten eines Modellelements an.**
  Nadeln werden in der Zentrale per Hand bewegt.

Objekte

In Java heißen die Elemente eines Modells **Objekte**.

Kundendienstmodell	Java-Modell
Die 43 Kundendienstmitarbeiter werden durch 43 Nadeln repräsentiert.	In Java würden die 43 Kundendienstmitarbeiter durch 43 Objekte modelliert.
Kunden werden durch Fähnchen markiert.	In Java würden spezielle Kunden-Objekte verwendet.
Wenn ein Kunde anruft, wird in der Zentrale ein Fähnchen in die Karte gesteckt.	In Java würde ein Kunden-Objekt erzeugt.

Verhalten

Alle Kundendienstmitarbeiter verhalten sich ähnlich: Sie fahren von Kunde zu Kunde und führen dort jeweils ihre Aufträge aus.

In Java haben alle Objekte für Kundendienstmitarbeiter das gleiche Verhalten.

Auch Kundenobjekte haben ein gleiches Verhalten: Nach Auftragseingang werden sie erzeugt. Sie werden gelöscht, sobald der Auftrag ausgeführt ist.

In Java wird das **Verhalten eines Objektes durch ein Programmstück definiert**, welches als **Klasse** bezeichnet wird.

Klassen und Instanzen

- **Kategorien von Elementen eines Modells heißen in Java Klassen.**
- Die **Definition einer Klasse** ist ein **Programmstück** (Code), welches spezifiziert, wie sich die **Objekte dieser Klasse verhalten**.
- Nachdem eine **Klasse definiert** wurde, **können Objekte** dieser Klasse **erzeugt werden**.
- **Jedes Objekt gehört zu genau einer Klasse und wird Instanz dieser Klasse genannt.**

Vordefinierte Klassen

- Glücklicherweise müssen Programmierer das Rad nicht ständig neu erfinden.
- Java beinhaltet ein große Anzahl **vordefinierter Klassen** und Objekte.
- Diese Klassen können verwendet werden, um auf einfache Weise **komplizierte Anwendungen** und z.B. **graphische Benutzeroberflächen** zu realisieren.

Ein einfaches Beispiel: Der Monitor

- Java Programme können Text auf den Monitor ausgeben
- In Java wird der Monitor durch ein Objekt modelliert
- Man kann an dieses Objekt **Nachrichten** senden
- Eine solche Nachricht enthält den auszugebenden Text

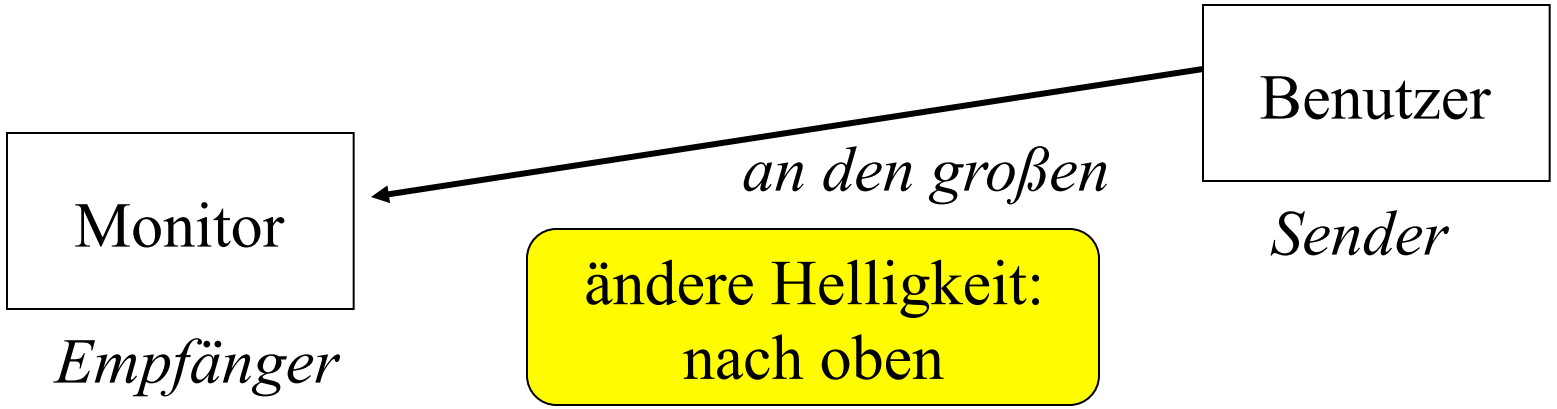
Ein einfaches Beispiel für eine Nachricht

Zwei Personen sitzen vor zwei unterschiedlich großen Bildschirmen. Eine Person sagt: „Würdest Du bitte die Helligkeit des großen erhöhen?“

Informationen in dieser Frage:

- des großen
- die Helligkeit soll verstellt werden
- nach oben

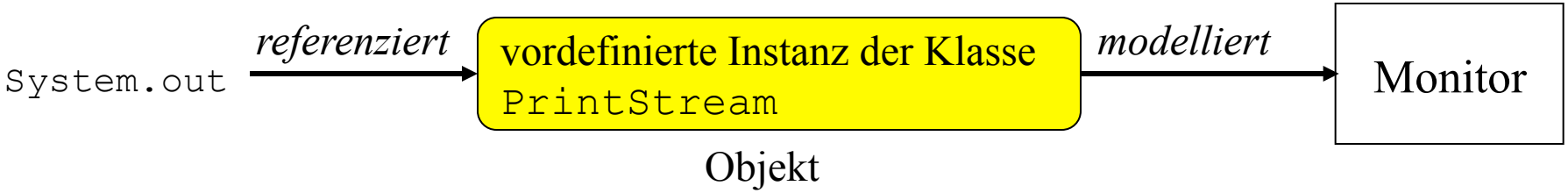
Die Aktion in der Java-Terminologie



- „Des großen“ ist eine **Referenz** , an die eine Nachricht (**Message**) geschickt wird.
- Die Nachricht spezifiziert ein **Verhalten** (Helligkeit verstellen)
- und enthält **weitere Details** (nach oben).

Repräsentation des Bildschirms in Java

- In Java wird der **Bildschirm** durch ein **vordefiniertes Objekt** repräsentiert.
- Dieses Objekt ist Instanz der Klasse **`PrintStream`**.
- Objekte der Klasse **`PrintStream`** erlauben das Ausgeben von **Zeichenketten**.



- Eine **Referenz in Java** ist eine Phrase, die einen **Bezug zu einem Objekt** herstellt. Wir sagen, dass sie das **Objekt referenziert**.
- **`System.out`** referenziert ein Objekt der Klasse **`PrintStream`**, welches den Monitor modelliert. **`System.out`** ist eine **Referenz** auf das Objekt.

Nachrichten/Messages in Java

- Um einen Text auf dem Bildschirm auszugeben, schickt man in Java eine **Nachricht** an das durch Objekt `System.out` referenzierte Objekt.
- Dieses Message spezifiziert ein **Verhalten**, welches die Klasse `PrintStream` zur Verfügung stellt.
- Hier ist diese Nachricht das Ausgeben einer Zeile, welche `println` genannt wird.
- Die **weiteren Details** sind dabei die Zeichen, die ausgegeben werden sollen, z.B.: „Welcome to Java!“.

Zusammenfassend:

- Referenz auf den Monitor: `System.out`
- Nachricht: `println`
- Notwendige Details: `("Welcome to Java!")`

Verschicken einer Nachricht an das `System.out`-Objekt

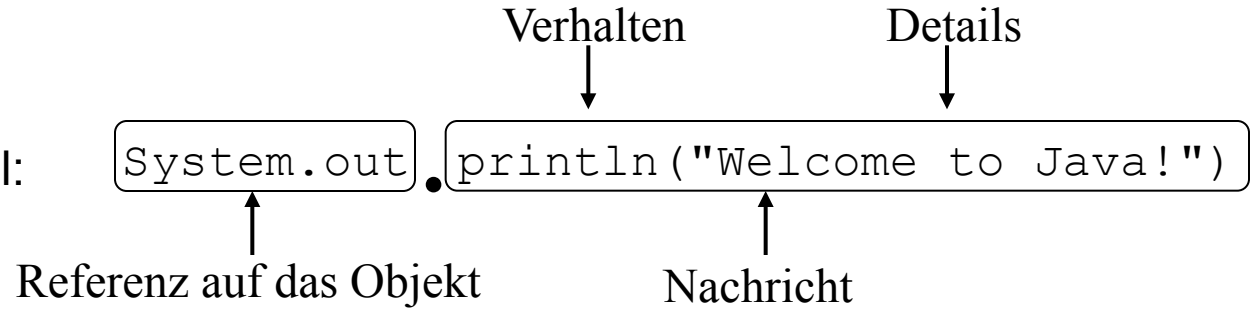
In Java bestehen Nachrichten aus folgenden Komponenten:

- Der Name des Verhaltens (z.B. `println`)
- Die weiteren Informationen (z.B. `"Welcome to Java!"`), die in Klammern eingeschlossen wird.

In Java verwenden wir Phrasen, die aus folgenden Komponenten bestehen, um eine Nachricht an den entsprechenden **Empfänger** zu senden:

- einer Referenz auf das Empfängerobjekt (z.B. `System.out`) gefolgt von einem Punkt und
- der Nachricht, die wir verschicken wollen.

Unser Beispiel:



Java Statements

- Das Versenden einer Nachricht an ein Objekt ist eine vom Programmierer spezifizierte Aktion.
- Diese Aktion wird vom Computer ausgeführt, wenn das Programm läuft.
- In Java werden alle Aktionen durch **Statements** spezifiziert.

Um ein Statement zu erzeugen, welches unsere Ausgabe-Aktion durchführt, schreiben wir die entsprechende Nachricht hin und fügen ein Semikolon hinzu:

```
System.out.println("Welcome to Java!");
```

Ein Java-Programm

Ziel: Ausgabe der beiden Zeilen

```
This is my first Java program
but it won't be my last.
```

Wir wissen:

- Es gibt ein vordefiniertes Objekt (referenziert durch `System.out`), dessen Verhalten auch das Ausgeben von Zeichen auf dem Bildschirm einschließt.
- Dieses Verhalten wird aktiviert, indem wir diesem Objekt eine Nachricht mit dem Namen `println` schicken.
- Die weiteren Details sind die **Argumente**, die das Objekt benötigt, um das Verhalten auszuführen (hier die Zeile, die ausgegeben werden soll).

Um zwei Zeilen auszugeben, schicken wir einfach zwei Nachrichten:

```
System.out.println("This is my first Java program");
System.out.println("but it won't be my last.");
```

Das komplette Programm

Um ein **gültiges Programm** zu erhalten, müssen wir seinen **Namen** festlegen und zusätzlichen Text hinzufügen. Wir nennen Namen **Identifier** oder **Bezeichner**.

Ein **Identifier** ist eine **Folge von Buchstaben, Ziffern und Unterstrichen**. Das **erste Zeichen** muss ein **Buchstabe** sein.

Beispiele: `Program1, x7, xyp, xrz`

Dies ergibt das bereits bekannte Programm:

```
class Program1 {
    public static void main(String[] arg) {
        System.out.println("This is my first Java program");
        System.out.println("but it won't be my last.");
    }
}
```

Einige Regeln (1)

Identifier: Klassen und Nachrichten müssen einen Bezeichner haben. Dabei wird zwischen Groß- und Kleinschreibung unterschieden.

Keywords: **Keywords** oder **Schlüsselworte** sind spezielle Worte mit einer vordefinierten Bezeichnung, wie z.B. `static, class, ...`

Reihenfolge von Statements: Statements werden in der Reihenfolge, ausgeführt, in der sie im Programm stehen.

```
System.out.println("This is my first Java program");
System.out.println("but it won't be my last.");
```

erzeugt eine andere Ausgabe als

```
System.out.println("but it won't be my last.");
System.out.println("This is my first Java program");
```

Einige Regeln (2)

Formatierung: Halten Sie sich bei der Erstellung von Java-Programmen an folgende Regeln:

1. Jede Zeile enthält höchstens ein Statement.
2. Verwenden Sie die Space oder Tabulator-Taste, um Statements einzurücken.
3. Halten Sie sich an den Stil dieser Vorlesung (siehe Zettel).

Vorteile:

1. Programme sind leichter lesbar,
2. leichter zu verstehen und damit auch
3. leichter zu warten.

Einige Regeln (3)

Prinzipiell ist Java sehr flexibel.

Eine der Hauptregeln: **Bezeichner müssen durch ein Leerzeichen getrennt sein.**

`classProgram1` entspricht nicht `class Program1`

Zulässig wäre aber z.B.:

```
class Program1 { public static void
main(String[] arg) { System.out.println("This is my first
Java program"); System.out.println("but it won't be my
last."); } }
```

Siehe: examples/Program1jammed.java

Einige Regeln (4)

Kommentare: Java erlaubt dem Programmierer, Kommentare in den Code einzufügen. Es gibt zwei Arten:

1. Eingeschlossene Kommentare, d.h. Text, der zwischen `/*` und `*/` eingeschlossen ist:

```
/*
 * This program prints out several greetings
 */
```

2. Zeilenkommentare: Alle Zeichen hinter einem `//` werden als Kommentare angesehen:

```
// This program prints out several greetings
```

Hinweise zur Verwendung von Kommentaren

1. Vor jeder Zeile, die mit dem Wort `class` beginnt, sollte ein Kommentar stehen, der die Klasse erklärt.
2. Kommentare sollten dazu dienen, Dinge zu erklären, die nicht unmittelbar aus dem Code abgelesen werden können.
3. Kommentare sollten nicht in der Mitte von einem Statement auftauchen.

Ein Programm mit Kommentaren

```
/*
 * This program prints my first Java experience and my
 * intent to continue.
 */
class Program1 {
    public static void main(String[] arg) {
        System.out.println("This is my first Java program");
        System.out.println("but it won't be my last.");
    }
}
```

Verwendung von `PrintStream`-Objekten

Wenn wir die Nachricht

```
println("something to display")
```

an das durch `System.out` referenzierte `PrintStream`-Objekt senden, führt das zur Ausgabe des entsprechenden Textes auf dem Bildschirm.

Jedes weitere Zeichen danach wird am Beginn der nächsten Zeile ausgegeben.

Die Nachrichten `print` und `println`

Alternative:

```
print("something to display")
```

Bei Verwendung von `print` erscheint das nächste gedruckte Zeichen in der gleichen Zeile:

```
System.out.print("JA");

System.out.print("VA");
```

ergibt die Ausgabe

```
JAVA
```

Eine Variante der `println`-Nachricht ist `println()`. Die Nachricht

```
System.out.println();
```

bewirkt, dass nachfolgende Ausgaben in der nächsten Zeile beginnen.

Ein weiteres Programm

```
/*
 * This program illustrates the use of print vs. println.
 */
class Program1print {
    public static void main(String[] arg) {
        System.out.print("This is my first Java program");
        System.out.print(" ");
        System.out.println("but it won't be my last.");
    }
}
```

Vom Programm zur Ausführung

Um ein Programm auf einem Rechner ausführen zu können, müssen wir

1. das **Programm für den Computer zugänglich machen**,
2. das **Programm** in eine Form **übersetzen**, die der Rechner versteht, und
3. den Computer anweisen, das **Programm auszuführen.**

Schritt 1: Eingeben des Programms

Um ein Programm für den Rechner zugänglich zu machen, müssen wir eine **Programmdatei** erstellen, d.h. eine **Textdatei, die den Programmtext enthält**.

In Java muss der **Name der Datei** den Namen `X.java` haben, wobei `X` der Programmname oder Klassenname ist.

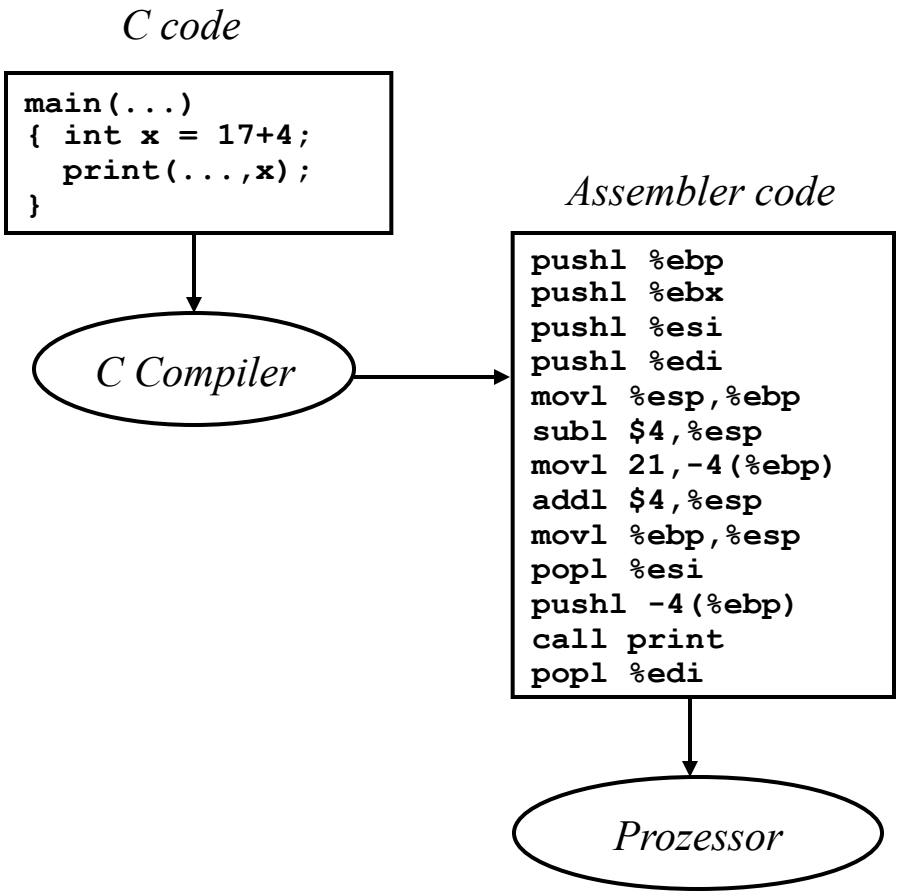
Es ist Konvention, dass Klassennamen groß geschrieben werden.

Schritt 2: Übersetzen des Programms

- Computer können Java-Programme nicht direkt ausführen.
- Computer sind nur in der Lage, Anweisungen einer primitiven und so genannten *Maschinensprache* auszuführen.
- Da wir eine abstrakte *Hochsprache* verwenden, müssen wir eine Möglichkeit finden, unsere Programme in Maschinen-Code zu überführen.
- Dafür gibt es verschiedene Modelle: Am weitesten verbreitet sind *Compiler* und *Interpreter*.

Das Compiler-Modell

Das Programm wird direkt in **Maschinencode** übersetzt und kann vom Rechner **unmittelbar ausgeführt** werden.



Das Interpreter-Modell

Prinzip:

- Das Programm wird von dem **Interpreter** direkt ausgeführt.
- Der Interpreter verwendet für jedes Statement bei der Ausführung die entsprechende Sequenz von Maschinenbefehlen.

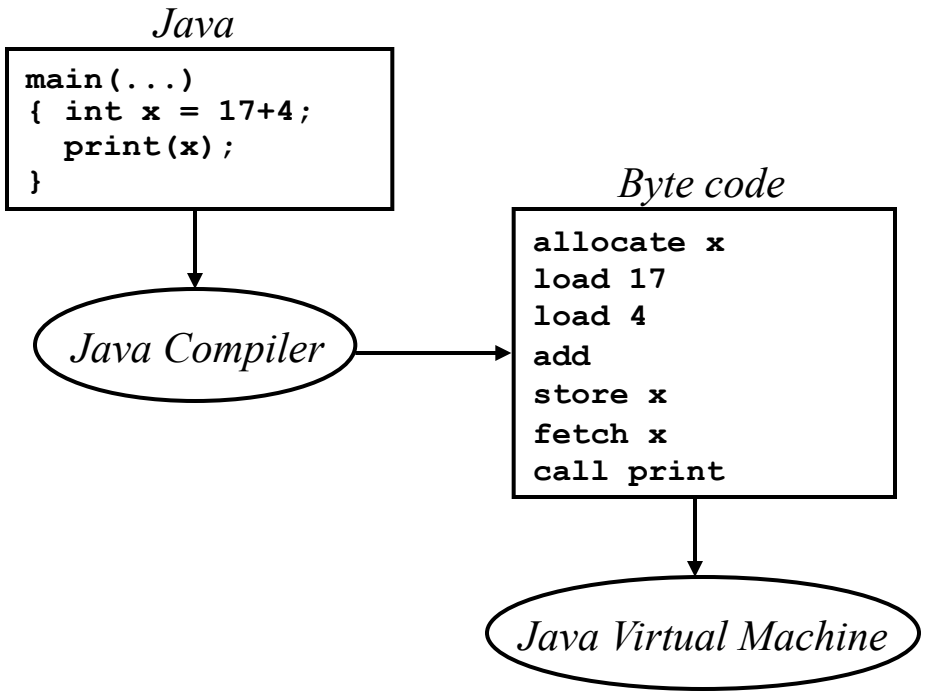
Vergleich zum Compiler:

1. Das Programm muss nicht erst übersetzt werden.
2. Die Interpretation ist langsamer als die Ausführung kompilierter Programme.

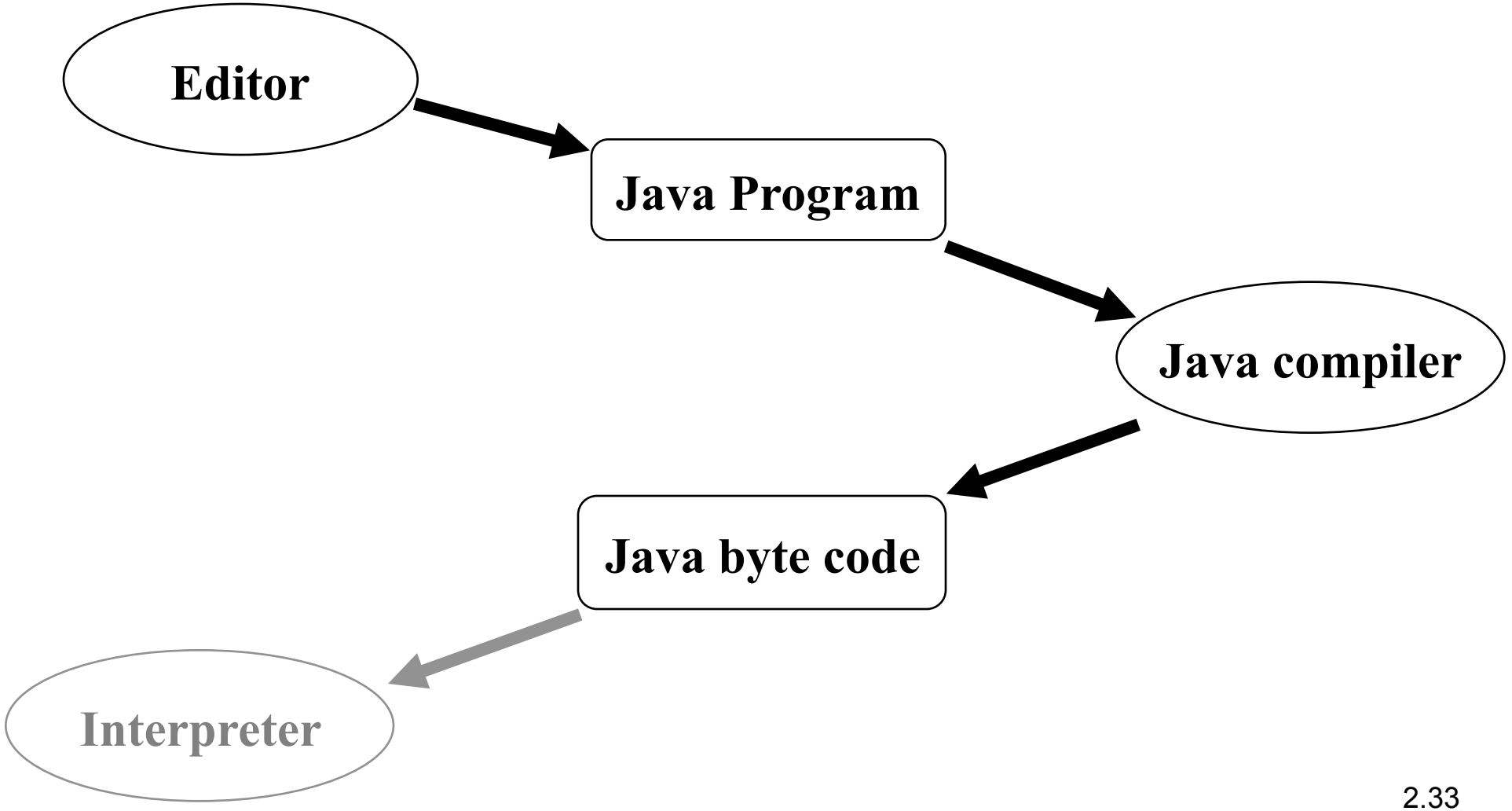
Das Java-Modell

Java verwendet einen Interpreter und einen Compiler:

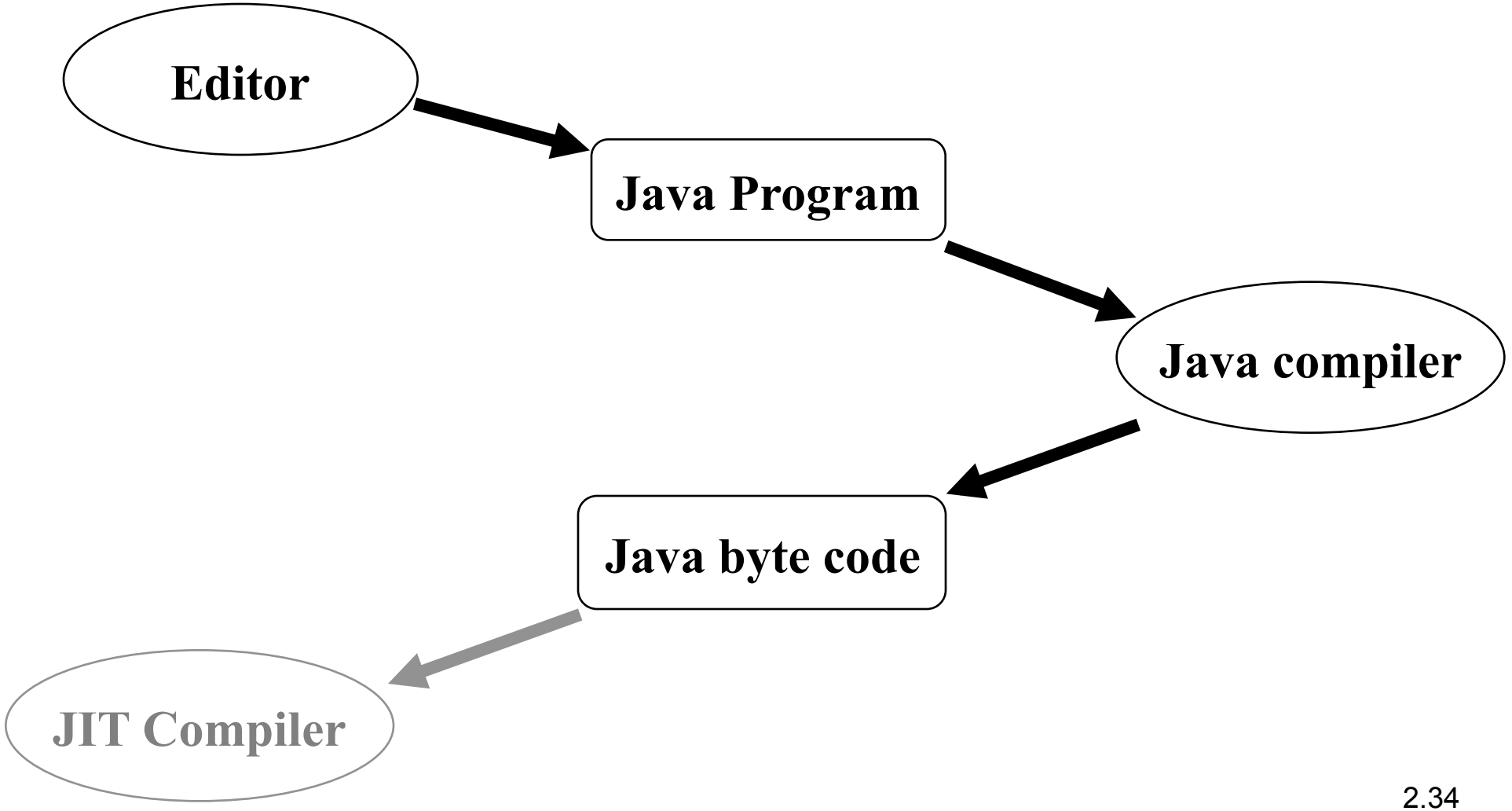
- Java **übersetzt** zunächst in eine **Byte Code** genannte Zwischensprache.
- Der Byte Code wird dann von der **Java Virtual Machine interpretiert**.



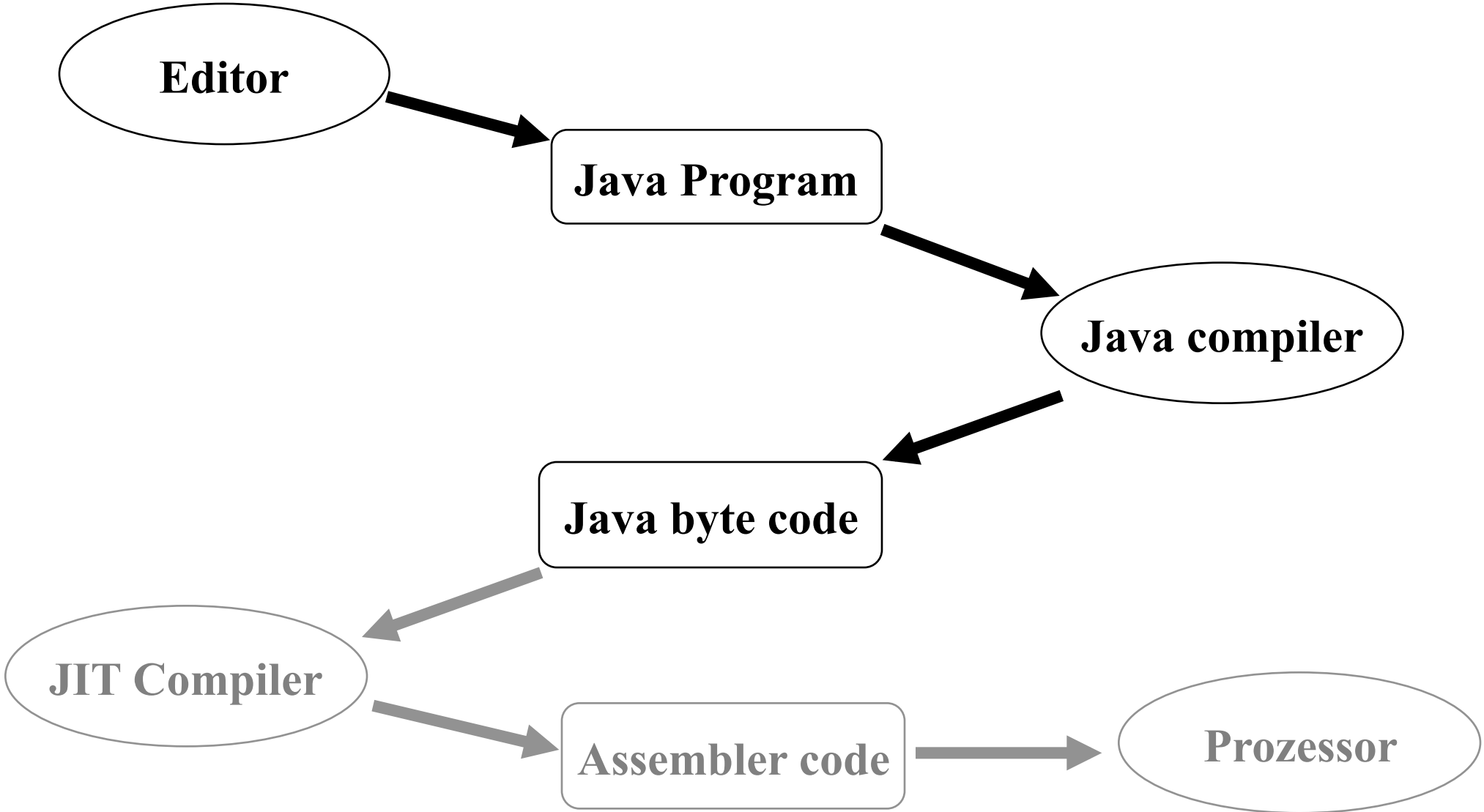
Erzeugung eines Ausführbaren Java-Programms



Erzeugung eines Ausführbaren Java-Programms



Erzeugung eines Ausführbaren Java-Programms



Programmierfehler

Wir unterscheiden grob zwei Arten von Fehlern:

1. **Compile-Zeit-Fehler** (Compile-Time Errors) werden bei der Übersetzung entdeckt und führen zum Abbruch des Übersetzungsprozesses.
2. **Laufzeitfehler** (Run-Time Errors) treten erst bei der Ausführung auf und liefern falsche Ergebnisse oder führen zum Absturz des Programms.

   Laufzeitfehler sind **häufig schwieriger zu entdecken**!

Beispiel für einen Compile-Time Error

```
// This will lead to errors
class Program1 {
    public static void main(String[] arg) {
        Sistem.out.println("This is my first Java program");
        Sistem.out.println("but it won't be my last.");
    }
}
```

Siehe: examples/Program1sistem.java

Ausgabe des Compilers

```
javac Program1sistem.java
Program1sistem.java:4: cannot resolve symbol
symbol  : class out
location: package Sistem
        Sistem.out.println("This is my first Java program");
                ^
Program1sistem.java:5: cannot resolve symbol
symbol : class out
location: package Sistem
        Sistem.out.println("but it won't be my last.");
                ^
2 errors

Compilation exited abnormally with code 1 at Mon Dec 10 20:30:09
```

Beispiel für einen Run-Time Error

```
class Program1last {
    public static void main(String[] arg) {
        System.out.println("This is my first Java program");
        System.out.println("but it will be my last.");
    }
}
```

Das Programm kann compiliert und ausgeführt werden und liefert die Ausgabe:

```
This is my first Java program
but it will be my last.
```

Zusammenfassung (1)

- **Programme** sind Texte, die ein Computer ausführt, um eine bestimmte Aufgabe durchzuführen.
- **Java-Programme** können aufgefasst werden als Modelle, die Objekte und ihr Verhalten beschreiben.
- **Objekte** sind in Java die **grundlegenden Modellierungskomponenten**.
- **Objekte** mit gleichem Verhalten werden in **Kategorien**, bzw. **Klassen** zusammengefasst.
- Das **Verhalten** eines Objektes wird durch das **Senden einer Nachricht** an dieses Objekt angestoßen.

Zusammenfassung (2)

- Die **Verhalten** der Objekte werden durch **Programmstücke** beschrieben, die aus einzelnen **Statements** bestehen.
- Eine typische Form eines **Statements** wiederum ist das **Versenden einer Nachricht** an ein Objekt.
- Um einem Objekt eine Nachricht zu senden, verwendet man eine **Referenz** auf dieses Objekt und den entsprechenden **Methodennamen**.
- Java-Programme werden in **Java Byte Code** übersetzt.
- Dieser Byte Code wird dann von dem **Java-Interpreter** auf dem Rechner ausgeführt.
- Zur Effizienzsteigerung werden in der Praxis just-in-time (JIT) Compiler eingesetzt, die Java Byte Code in Assembler Code übersetzen.