Buch, Kapitel 2: Klassendefinitionen

Java Quellcode

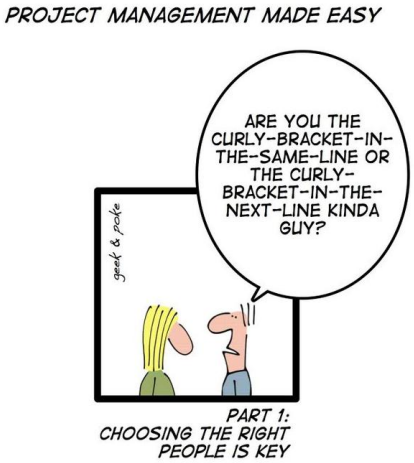
J. Kleimann, H.-W. Sehring, D. Versick, F. Zimmermann

Studiengang Wirtschaftsinformatik

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

Vorlesungsinhalt

1. Lerninhalte
2. Exemplarvariablen
3. Konstruktoren
4. Methoden
5. Alternativen
6. Lokale Variablen



Lerninhalte

Nach Beenden dieser Lektion werden Sie

- Informationen in Exemplarvariablen und lokalen Variablen hinterlegen und abrufen können.
- Objekte mit Konstruktoren erzeugen können.
- Methoden zum Speichern, Verändern und Abfragen von Informationen verstehen und schreiben können.
- Informationen zwischen Methoden in Form von Parametern übergeben können.
- Zuweisungsoperationen verstehen und nutzen können.

Ein Fahrkartenautomat: externe Sicht

Die Lerninhalte werden an folgendem Anwendungsbeispiel formuliert:

Im Beispiel „Naive Ticket Machine“ wird ein Fahrkartenautomat simuliert.

- Der Automat gibt Tickets für einen festen Preis aus. Wie wird dieser Preis festgelegt?
- Wie wird das Geld in den Automaten gesteckt?
- Wie ermittelt der Automat, wie viel Geld schon hineingesteckt wurde?

Ein Fahrkartenautomat: interne Sicht

- Mit Objekten kann man kommunizieren. Diese Kommunikation beschreibt das **Verhalten** der Objekte.
- Um zu verstehen, wie dieses Verhalten entsteht, muss man in die Klassendefinition, insbesondere in die dort definierten **Methoden** hineinschauen.
- **Exemplarvariablen, Methoden und Konstruktoren** sind die Grundbausteine aller Java Klassen.

Aufbau einer Klasse

Interne Bausteine einer Klasse

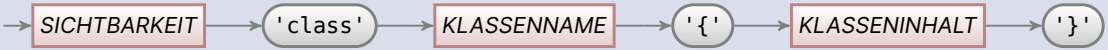
```
/**
 * Einfacher Fahrkartenautomat
 *
 * @author Mustermann
 */
public class TicketMachine {
  //Exemplarvariablen
  ...
  //Konstruktoren
  ...
  //Methoden
  ...
}
```

- **Kommentare**: /** ...*/, /* ...*/ und // ...
- Reservierte Worte: **`public`**, **`private`**, **`class`**, **`if`**, **`final`**, ...
- Bezeichner für Klassen, Exemplarvariablen und Methoden: `TicketMachine`, `price`, `getPrice`
- Spezielle Symbole wie +, - , =, (,), ...

Klassendefinition (Syntax)

Einfache Angabe der Sprachsyntax durch **Syntaxdiagramme**.

Verallgemeinert (Klassendefinition)



Am Beispiel (Klassendefinition)

Übung

Macht es einen Unterschied, ob wir **`public class`** `TicketMachine` oder **`class public`** `TicketMachine`
in der äußeren Klammer einer Klasse schreiben?

- Editieren Sie den Quelltext der Klasse `TicketMachine` auf diese Weise und schließen Sie dann das Editorfenster. Stellen Sie einen Unterschied im Klassendiagramm fest?
- Welche Fehlermeldung erhalten Sie, wenn Sie nun den Knopf ÜBERSETZEN klicken? Glauben Sie, dass die Fehlermeldung klar erläutert, was falsch ist?
- Nehmen Sie Ihre Änderung wieder zurück und achten Sie darauf, dass der Fehler nach dem Übersetzen nicht wieder auftaucht.

Exemplarvariablen (Datenfelder)

- Speichern Informationen eines Objekts.
- Definieren den Objektzustand.
- Können mit BlueJ inspect erkundet werden.
- Einige können sich häufig, andere nie ändern (**`final`**).
- Sichtbarkeit: **`private`**(, **`public`**)
- In Java müssen Variablen deklariert werden.
- Typ: **`int`**
- Bezeichner: price

Beispiel

```
public class TicketMachine {
  private int price;
  private int balance;
  private int total;

  //Weitere Details ausgelassen
}
```

Exemplarvariablen (Syntax)

Datenfelder, Instanzvariablen, Eigenschaften

Verallgemeinert

Am Beispiel

Übung

2.12

Welchen Typ haben jeweils die folgenden Datenfelder Ihrer Meinung nach?

```
private int counter;
private Student speaker;
private Server central;
```

2.13

Was sind die Namen der folgenden Datenfelder?

```
private boolean alive;
private Person tutor;
private Game game;
```

2.14

Beurteilen Sie, aufbauend auf Ihren Kenntnissen der Namenskonventionen für Klassen, welche der Typnamen in den Übungen 2.12 und 2.13 Klassennamen sind.

Übung

2.15

Ist es bei der folgenden Deklaration eines Datenfeldes aus der Klasse `TicketMachine`

```
private int price;
```

wichtig, in welcher Reihenfolge die drei Wörter stehen?
Editieren Sie die Klasse `TicketMachine` und probieren Sie verschiedene Reihenfolgen aus. Gibt Ihnen das Aussehen des Klassendiagramms nach jeder Änderung einen Hinweis, ob andere Reihenfolgen möglich sind?
Überprüfen Sie durch Klicken auf ÜBERSETZEN, ob es eine Fehlermeldung gibt. Stellen Sie sicher, dass nach Ihren Experimenten die Originalversion wiederhergestellt wird!

Übung

2.16

Ist es zwingend erforderlich, dass am Ende einer Datenfelddeklaration ein Semikolon steht? Experimentieren Sie auch hier mit dem Editor. Die Regel, die Sie dabei lernen, ist sehr wichtig, deshalb sollten Sie sich diese einprägen.

2.17

Schreiben Sie die vollständige Deklaration eines Datenfeldes mit dem Typ **`int`** und dem Namen `status`.

Acht primitive Datentypen zum Rechnen

Name	Größe	Wertebereich	Beschreibung
`boolean`	8 bit	true / false	Boolescher Wahrheitswert
`char`	16 bit	z. B. 'A'	Unicode (UTF-16)
`byte`	8 bit	-128 ... 127	Zweierkomplement-Wert
`short`	16 bit	-32.768 ... 32.767	Zweierkomplement-Wert
`int`	32 bit	±2 Milliarden	Zweierkomplement-Wert
`long`	64 bit	±9 Trillionen	Zweierkomplement-Wert
`float`	32 bit	±1,4E-45...±3,4E+38	Gleitkommazahl (IEEE 754)
`double`	64 bit	±4,9E-324...±1,7E+308	doppelt genau (IEEE 754)

- **In kommerziellen Anwendungen verwendet man KEINESFALLS `float` oder `double` für Geldbeträge.**

Zahlenliterale

Ganzzahlige Literale

Literal	Dezimalwert	Datentyp	Beschreibung
123	123	**`int`**	dezimal
0123	83	**`int`**	oktal
0×123	291	**`int`**	hexadezimal
123L	123	**`long`**	dezimal
0xbadFacel	195951310	**`long`**	hexadezimal

Gleitpunktliterale

Literal	Datentyp
1.23f	**`float`**
.23F	**`float`**
1.23	**`double`**
123e-2	**`double`**
.123E1	**`double`**

Konstruktoren

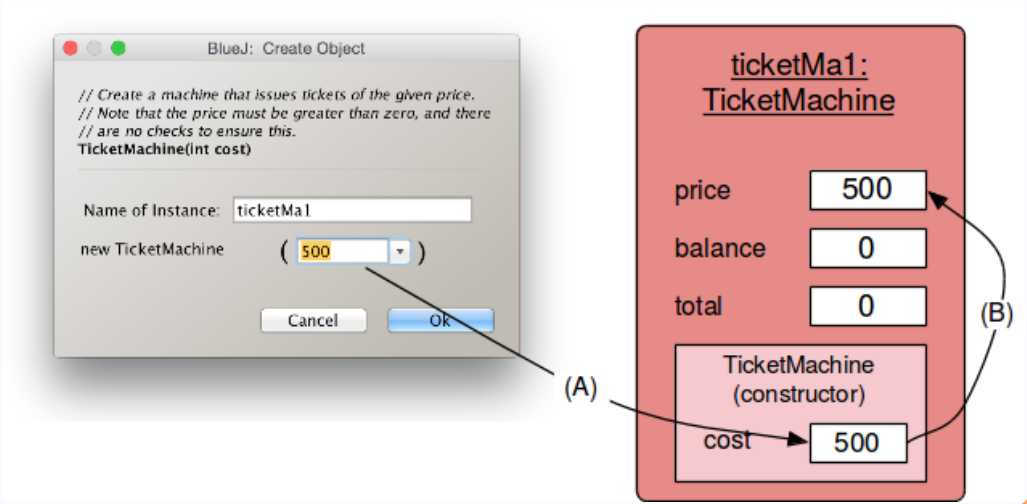
Konstruktoren sind spezielle Programmteile. Sie ...

- erzeugen Objekte. Dabei wird automatisch Speicherplatz für die Exemplarvariablen reserviert.
- initialisieren ein Objekt, so dass es benutzt werden kann.
- weisen den Exemplarvariablen Initialwerte zu.
- haben immer denselben Namen wie die Klasse.

Beispiel

```
public TicketMachine(int fare) {
  price = fare;
  balance = 0;
  total = 0;
}
```

Informationsübergabe durch Parameter



Konstruktor (Syntax)

Verallgemeinert (Konstruktor)

Am Beispiel (Konstruktor)

Zuweisungen

- Werte können in Exemplarvariablen mittels **Zuweisungen** gespeichert werden.
- Eine Variable speichert nur einen Wert. Durch eine Zuweisung geht der vorhergehende Wert verloren.
- Ein Semikolon beendet die Zuweisung.

- Das generelle Schema:
  `variable = ausdruck;`
- Beispiel
  `balance = balance + amount;`

Zuweisungs Operatoren/Ausdrücke

Die Programmiersprache Java hat aus der Programmiersprache C einige abkürzende Schreibweisen übernommen:

Anweisung	Bedeutung	Erklärung
$x = y;$	$x = y;$	Einfache Zuweisung
x += $y;$	$x = x + y;$	Zuweisungsoperator
x <op>= $y;$	$x = x$ <op> $y;$	<op> ist beliebiger Operator
$y = x++;$	$y = x; x = x + 1;$	Postinkrement
$y = x--;$	$y = x; x = x - 1;$	Postdekrement
$y = --x;$	$x = x - 1; y = x;$	Prädekrement
$y = ++x;$	$x = x + 1; y = x;$	Präinkrement

Bezeichner für Variablen

- Gute Bezeichner sind Grundlage jedes lesbaren Programms.
- Verwenden Sie aussagekräftige, englische Namen:

```
price, amount, name, age, ...
```

- Variablenbezeichner beginnen klein und werden in CamelCase geschrieben:

```
bookTitle, ...
```

- Vermeiden Sie Obfuskation:

```
~~w, t5, xyz123~~
```

Methoden

- Methoden sind die interne Beschreibung des Verhaltens eines Objekts.
- Methoden haben eine Signatur und einen Rumpf.
- Business Methode(n)/Geschäftsmethode(n)/Geschäftslogik(en)/Fachliche Methode beschreib(t/en) die Verantwortung einer Klasse. Sie geben die Begründung für den Klassennamen.
- Spezielle Methoden
  - *getter*-Methoden geben Auskunft über den Zustand eines Objekts.
  - *setter*-Methoden erlauben es den Zustand eines Objekts zu ändern.

Methodenbestandteile

Signatur

- Die Signatur beschreibt die externe Sicht
  - Beispiel: `public int getPrice()`
  - Die Signatur beschreibt:
    - die **Sichtbarkeit**: `public`
    - den Ergebnistyp des Aufrufs: `int`
    - Methodenname: `getPrice`
    - Parameterliste: `()` — hier leer.

Methodenrumpf

- Der Methodenrumpf beschreibt die interne Sicht
  - ist ein Block …
  - enthält Anweisungen, z.B.
    - Deklarationen
    - Zuweisungen
    - Bedingte Anweisungen
    - Schleifen
    - ggf. return-Anweisung(en)

Beispiel getter-Methode (Sondierende Methode)

Methode getPrice

```
public int getPrice() {
  return price;
}
```

- visibility modifier (**`public`**)
- return type (**`int`**)
- Methodenname/-bezeichner (`getPrice`)
- Parameterliste ()
- **`return`** Anweisung (**`return`** price)
- Start Methodenrumpf {
- Ende Methodenrumpf }

getter-Methode (Syntax)

Verallgemeinert (*getter*-Methode), Datenfeld `xyz`

Am Beispiel (*getter*-Methode), Datenfeld `price`

getter-Methode einer Exemplarvariablen

- *getter*-Methoden heißen `getXyz`, wobei `xyz` der Name einer Exemplarvariablen ist. Bei boolschen Variablen (**`true`**, **`false`** Werte) heißen sie auch `isXyz`.
- Eine *getter*-Methode hat immer einen Rückgabedatentyp, der nicht **`void`** ist.
- Eine *getter*-Methode muss ein Ergebnis des Rückgabedatentyps aus der Signatur an seinen Aufrufer zurückgeben.
- Eine *getter*-Methode hat immer ein **`return`** statement.

- **`return`** ist nicht drucken/ausgeben, sondern **zurückgeben**.

Test: Was ist falsch

Finden Sie fünf Fehler

```
public class CokeMachine {
  private price;

  public CokeMachine() {
    price = 300
  }

  public int getPrice {
    return Price;
  }
```

setter-Methoden (Verändernde Methode)

- Private Exemplarvariablen haben häufig *setter*-Methoden, die den Wert ändern.
- Der Name der *setter*-Methode für die Exemplarvariable `xyz` ist `setXyz`.
- *setter*-Methoden haben einen einzigen formalen Parameter, der den neuen Wert übergibt.
- Im Rumpf der Methode steht eine Zuweisung an die Exemplarvariable.

Beispiel setter-Methode für discount Exemplarvariable

Methode setDiscount

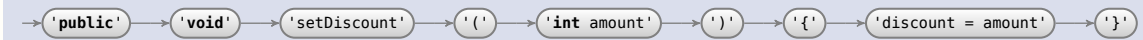
```
public void setDiscount(int amount) {
  discount = amount;
}
```

- visibility modifier **`public`**
- return type **`void`**
- method name setDiscount
- Parameterlist: **`int`** `amount`
- keine Rückgabeanweisung (**`return`**)

setter-Methode (Syntax)

Verallgemeinert (*setter*-Methode, Datenfeld `xyz`)

Am Beispiel (setter Methode), Datenfeld `discount`



Kapselung

- *setter*-Methoden müssen die Exemplarvariable in jedem Fall verändern.
- Der Parameter kann auf Plausibilität überprüft werden. Ist der Wert nicht sinnvoll, wird die Exemplarvariable nicht verändert.
- *setter* schützen die Exemplarvariable vor fehlerhaften Eingaben.
- *setter* unterstützen Kapselung.

Fachliche Methoden

Verhaltensdefinierende Methode

Fachliche/Business Methoden/Geschäftslogik

- erfüllen die Aufgaben, die die Objekte erledigen sollen. Sie setzen den „raison d'être“ um.
- verändern in der Regel den Status eines Objekts aus einem konsistenten Zustand in einen konsistenten Zustand.
- Können ein Ergebnis zurückgeben oder haben den Ergebnistyp **`void`**.
- Betreffen in der Regel mehr als eine Exemplarvariable.
- machen ggf. Ausgaben.

Beispiel Fachliche Methode

printTicket

```
public void printTicket() {
  // Den Ausdruck eines Tickets simulieren.
  System.out.println("##################");
  System.out.println("# Die BlueJ-Linie");
  System.out.println("# Ticket");
  System.out.println("# " + price + " Cent.");
  System.out.println("##################");
  System.out.println();

  // Die Gesamtsumme mit dem eingezahlten Betrag akt...
  total = total + balance;
  // Die bisherige Bezahlung zurücksetzen.
  balance = 0;
}
```

Textausgabe mit Stringkonkatenation

- `System.out.println(4+5);`
  9
- `System.out.println(“ubun“+“tu“);`
  ubuntu
- `System.out.println(“Result: “+6);`
  Result: 6
- `System.out.println("# " + price + " cent");`

  # 500 cent

Quiz (Nicht schön, aber leider auch nicht selten)

Was ist die Ausgabe?

- `System.out.println(5 + 6 + “hello“);`
- `System.out.println(“hello“+ 5 + 6);`

Was wissen wir über Methoden?

- Methoden setzen das Verhalten der Objekte um.
- Eine Methode hat eine Sichtbarkeit, einen Rückgabedatentyp (Return Typ), einen Namen und eine Parameterliste
  - Der Return Typ kann **`void`** („nichts“) sein
  - Wenn der Return Typ nicht **`void`** ist, muss die Methode mit einem return Statement beendet werden. Der in der **`return`**-Anweisung angegebene Wert wird an den Aufrufer der Methode übergeben.
  - Über Parameter können zusätzliche Informationen an die Methode übergeben werden.

Übung

Untersuchen Sie die Klasse `TicketMachine`:

- Führen Sie eine statische Code-Analyse durch, d.h. untersuchen Sie den Quelltext und ordnen Sie die gelernten Konzepte entsprechend zu.
- Fallen Ihnen Schwachstellen im Aufbau des Ticketautomaten auf? Erfüllt dieser alle Funktionalitäten, die man von einem Ticketautomaten erwarten würde?
- Führen Sie den Code aus und versuchen Sie durch kreative Nutzung Fehlverhalten des Ticketautomaten zu provozieren.

Nachteile der naiven TicketMachine

Das Verhalten der TicketMachine lässt an mehreren Stellen zu wünschen übrig:

- Keine Plausibilitätsprüfung des eingeworfenen Betrags
- Kein Rückgeld
- Keine Plausibilität der Initialisierung.

Was kann man besser machen?

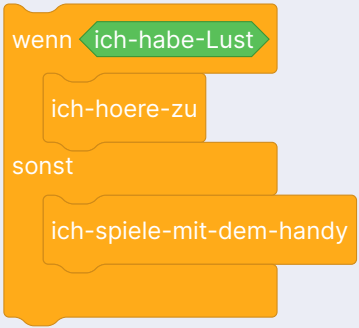
- Verschiedene Verarbeitungen abhängig von der Situation. Alternativen mit einer **`if`**-Anweisung.

Alternativen im täglichen Leben

Das Leben ist voller Entscheidungen:

- Wenn ich Lust habe, höre ich zu, was der Dozent sagt.
- Wenn nicht, dann spiele ich mit meinem Handy.

Alternative in Scratch



Alternative in Java

```
if (ich-habe-Lust)
  ich-höre-zu;
else
  ich-spiele-mit-dem-Handy;
```

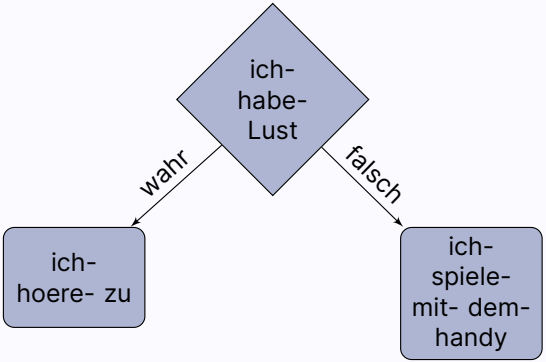
Alternative in Racket

```
(if ich-habe-Lust
    'ich-höre-zu
    'ich-spiele-mit-dem-Handy)
```

Alternativen

Zum Vergleich ...

ich-habe-Lust: true	ich-habe-Lust: false
ich-hoere-zu	ich-spiele-mit-dem-handy
	∅



Alternative in Pascal

```
IF ich-habe-Lust THEN
BEGIN
  ich-höre-zu;
END
ELSE
BEGIN
  ich-spiele-mit-dem-Handy;
END;
```

Alternative in Python

```
if ich-habe-Lust:
  ich-höre-zu
else:
  ich-spiele-mit-dem-Handy
```

Alternativen in Java

Alternativen (**Bedingte Anweisungen**) in Java werden mit einer **`if`**-Anweisung formuliert. Eine Bedingung (**Boolescher Ausdruck**) legt fest, ob der **`if`**- oder der **`else`**-Zweig ausgeführt wird.

if statement

```
if (perform some test) {
  Do these statements if the test gave a true result;
} else {
  Do these statements if the test gave a false result;
}
```

Bedingungen stehen bei **`if`** und **`while`** (später) in runden Klammern.

Plausiprüfung in fachlicher Methode

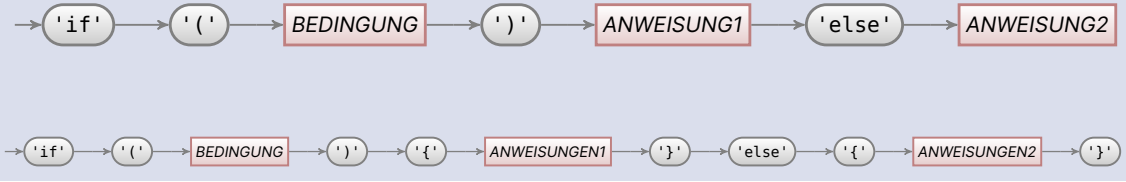
Nur positive Geldbeträge sollen akzeptiert werden. Andernfalls wird eine Fehlermeldung ausgegeben.

Prüfung des Parameters

```
public void insertMoney(int amount){
  if (amount > 0) {
    balance = balance + amount;
  } else {
    System.out.println("Bitte nur positive Beträge"
    + " verwenden: "+amount);
  }
}
```

Alternative/Bedingte Anweisung (Syntax)

Verallgemeinert (Bedingte Anweisung)



Vergleichsoperatoren

Operator	Bedeutung
==	Gleich
!=	Ungleichheit
<	Kleiner
>	Größer
<=	Kleiner oder gleich
>=	Größer oder gleich

- Ergebnisse eines Vergleichs können einer booleschen Variablen zugewiesen werden
- **Beachte** die Verwechslungsmöglichkeit == und = .
  Compiler wird Fehler anzeigen.

Übungen

2.47

Sagen Sie voraus, was passiert, wenn Sie die Prüfung in `insertMoney` so verändern, dass der Größer-gleich-Operator verwendet wird: `if (betrag >= 0)` Überprüfen Sie Ihre Vorhersage durch einige Tests. Welchen Einfluss hat diese Änderung auf das Verhalten der Methode?

2.48

Schreiben Sie die **`if`**-**`else`**-Anweisung der `insertMoney` Methode so um, dass sich die Methode zwar immer noch korrekt verhält, aber eine Fehlermeldung ausgegeben wird, wenn der boolesche Ausdruck wahr ist, beziehungsweise der bisher eingezahlte Betrag erhöht wird, wenn dieser falsch ist. Hierzu werden Sie offensichtlich die Bedingung umschreiben müssen.

Nicht zu empfehlen

- Bei nur einer Anweisung können die Blockklammern weggelassen werden. Dennoch ist hiervon abzuraten.
- Ein **`else`**-Zweig wird stets dem davor stehenden **`if`** zugeordnet.

```
if (x >= 0)
  if ( x > 0)
    System.out.println("größer als Null");
  else
    System.out.println("gleich Null");
```

```
if (x >= 0)
  if ( x > 0) System.out.println("größer als Null");
else
  System.out.println("kleiner Null?");
```

Strukturierte Anweisungen: Mehrfache Alternative

Switch-Anweisung

Anweisungsform

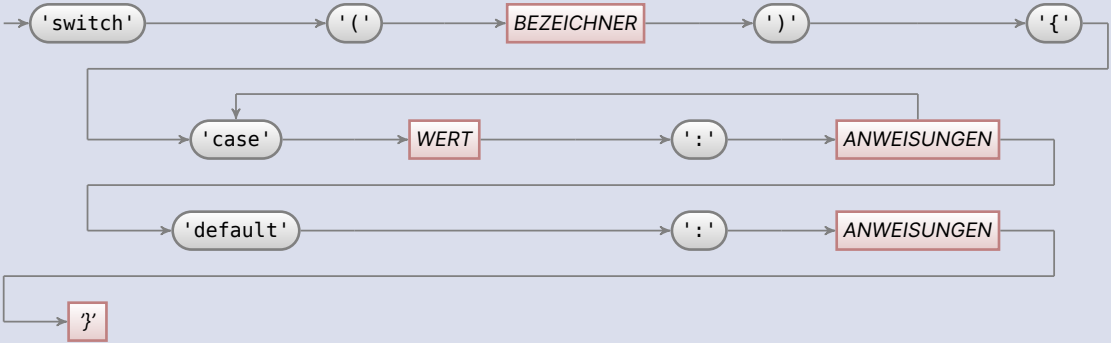
```
int x = 4;

switch(x) {
  case 3: System.out.println("Drei"); break;
  case 4: System.out.println("Vier"); break;
  case 5: System.out.println("Fünf"); break;
  default: System.out.println("was anderes");
}
```

- Achte auf **break** Anweisung
- Ausgabe mit **break**:
  Vier

- Ausgabe ohne **break**:
  Vier
  Fünf
  was anderes

Switch-Anweisung (Syntax)

Verallgemeinert (Switch-Anweisung)



Wiederholung Variablen

- Exemplarvariablen sind eine Art von Variablen.
  - Sie speichern Informationen über das Objekt und stellen diese über die gesamte **Lebensdauer** des Objekts zur Verfügung.
  - Die Lebensdauer eines Objekts fängt mit einem Konstruktoraufruf an und endet, wenn kein aktives Programmteil mehr auf das Objekt zugreifen möchte.
  - Sie können in allen Methoden einer Klasse genutzt werden.
- Parameter sind eine zweite Art von Variablen.
  - Parameter übergeben Werte von Nutzer einer Methode zur Methode.
  - Jede Nutzung einer Methode bekommt einen neuen Satz von Werten.
  - Parameter sind nur während der Ausführung einer Methode nutzbar und nicht darüber hinaus.

Wechselgeld herausgeben

Ein erfolgloser Versuch

```
public int returnChange() {
  // Gibt das verbleibende Geld zurück.
  return balance;
  // Setzt die aktuelle Summe auf 0.
  balance = 0;
}
```

Der Java Compiler lässt es nicht zu, nach einem **`return`** noch Anweisungen zu hinterlegen.
Fehlermeldung: Unreachable Code.

Lokale Variablen

Zum Berechnen von Zwischenergebnissen können Methoden **lokale Variablen** verwenden.

- Ähnlich Parametern können diese nur innerhalb einer Methode verwendet werden.
- Anders als bei Parametern erhalten lokale Variablen ihre Werte durch Zuweisungen innerhalb der Methode.
- Die Lebensdauer ist auf die Ausführung der Methode begrenzt.

Lokale Variablen

Beispiel returnChange

```
public int returnChange() {
  int change;
  change = balance;
  balance = 0;
  return change;
}
```

- Lokale Variablen haben keinen Sichtbarkeitsmodifikator.
- Sichtbarkeit im Block in dem sie definiert werden nach der Deklaration.
- Lebensdauer solang der Block ausgeführt wird.

Übung

Warum liefert die folgende Version von `returnChange` nicht das gleiche Ergebnis wie das Original?

```
public int returnChange() {
  balance = 0;
  return balance;
}
```

Blöcke als Gültigkeitsbereich

- Blöcke können an beliebigen Stellen stehen, wo auch Anweisungen stehen dürfen.
- Blöcke beginnen mit { und enden mit } .
- Variablen können lokal für diese Blöcke definiert werden. Blöcke sind ein Gültigkeitsbereich.

Gültigkeitsbereich

```
int x = 0;
{ // Das y gibt es nur in diesem Block
  int y = 4;
  x = y;
}
System.out.println(x); //y gibt es nicht mehr
```

Übung

Verbessern Sie nun den Ticketautomaten, dass dieser folgende Funktionalitäten unterstützt:

- Ungültige Geldeinwürfe werden abgelehnt
- Das Ticket kann nur dann gedruckt werden, wenn auch genügend Geld eingeworfen wurde
- Wechselgeld bzw. Rückgeld kann auf Anforderung zurückgegeben werden

Weitere Übungen

Schreiben Sie entsprechende Methoden, die folgende Anforderungen erfüllt:

- **`public void`** `isFailed(`**`float`** `note)` gibt auf der Konsole aus, ob ein Student mit der angegebenen Note durchgefallen ist.
- **`public void`** `printCipherAsNumeral(`**`char`** `cipher)` gibt auf der Konsole aus, wie eine Ziffer in natürlicher Sprache aussieht, z.B. `printCipherAsNumeral('0')` ⇒ `''Null''`.
- Extra: Implementieren Sie eine Klasse `NumberGuessing`, in der Sie im Konstruktor die gesuchte Zahl übergeben. Eine Methode `rate(...)` gibt jeweils einen Text wie: `''Leider etwas zu viel''` oder `''Viel zu wenig!''` aus.

Fragen?

Für weitere Fragen im Nachgang können Sie mich gerne über Moodle oder via E-Mail kontaktieren!

Konzepte: Zusammenfassung I

- **Kommentare** Kommentare werden im Quelltext einer Klasse angegeben, um menschlichen Lesern das Verstehen des Codes zu erleichtern. Sie haben keinen Einfluss auf die Funktionalität einer Klasse.
- **Datenfelder** Datenfelder speichern die Daten, die ein Objekt benutzt. Datenfelder werden auch als Instanzvariablen, Attribute oder Exemplarvariablen bezeichnet.
- **primitive Datentypen** Die primitiven Typen in Java sind die Typen, die keine Objekttypen sind. Die gebräuchlichsten primitiven Typen sind **`int`**, **`boolean`**, **`char`**, **`double`** und **`long`**. Primitive Typen haben keine Methoden.
- **Konstruktoren** Konstruktoren ermöglichen, dass ein Objekt nach seiner Erzeugung in einen gültigen Zustand versetzt wird.
- **Zuweisungen** Zuweisungen speichern den Wert auf der rechten Seite eines Zuweisungsoperators in der Variablen, die auf der linken Seite genannt ist.
- **Sichtbarkeit** Durch die Sichtbarkeit einer Variablen wird der Bereich innerhalb des Quelltextes definiert, indem eine Variable zugreifbar ist.
- **Sondierende Methode** Sondierende Methoden liefern Informationen über den Zustand eines Objekts.

Konzepte: Zusammenfassung II

- **Verändernde Methode** Verändernde Methoden ändern den Zustand eines Objekts.
- **Bedingte Anweisungen** Eine bedingte Anweisung führt eine von zwei Aktionen aus, abhängig vom Ergebnis einer Prüfung.
- **Boolescher Ausdruck** Boolesche Ausdrücke haben nur zwei mögliche Werte: wahr (**`true`**) und falsch (**`false`**). Sie werden oft verwendet, um die Auswahl zwischen zwei Ausführungspfaden in einer bedingten Anweisung zu treffen.
- **Switch-Anweisung** Eine switch-Anweisung wählt aus verschiedenen möglichen Anweisungsfolgen eine Folge zur Ausführung aus.
- **Lebensdauer** Die Lebensdauer einer Variablen legt fest, wie lange sie existiert, bevor sie zerstört wird.
- **lokale Variablen** Eine lokale Variable ist eine Variable, die innerhalb einer Methode deklariert und benutzt wird. Sie ist nur innerhalb der Methode zugreifbar und ihre Lebensdauer entspricht der ihrer Methode.

Kommentare

Kommentare werden im Quelltext einer Klasse angegeben, um menschlichen Lesern das Verstehen des Codes zu erleichtern. Sie haben keinen Einfluss auf die Funktionalität einer Klasse.

Datenfelder

Datenfelder speichern die Daten, die ein Objekt benutzt. Datenfelder werden auch als Instanzvariablen, Attribute oder Exemplarvariablen bezeichnet.

primitive Datentypen

Die primitiven Typen in Java sind die Typen, die keine Objekttypen sind. Die gebräuchlichsten primitiven Typen sind `int`, `boolean`, `char`, `double` und `long`. Primitive Typen haben keine Methoden.

Konstruktoren

Konstruktoren ermöglichen, dass ein Objekt nach seiner Erzeugung in einen gültigen Zustand versetzt wird.

Zuweisungen

Zuweisungen speichern den Wert auf der rechten Seite eines Zuweisungsoperators in der Variablen, die auf der linken Seite genannt ist.

Sichtbarkeit

Durch die Sichtbarkeit einer Variablen wird der Bereich innerhalb des Quelltextes definiert, indem eine Variable zugreifbar ist.

Sondierende Methode

Sondierende Methoden liefern Informationen über den Zustand eines Objekts.

Verändernde Methode

Verändernde Methoden ändern den Zustand eines Objekts.

Bedingte Anweisungen

Eine bedingte Anweisung führt eine von zwei Aktionen aus, abhängig vom Ergebnis einer Prüfung.

Boolescher Ausdruck

Boolesche Ausdrücke haben nur zwei mögliche Werte: wahr (`true`) und falsch (`false`). Sie werden oft verwendet, um die Auswahl zwischen zwei Ausführungspfaden in einer bedingten Anweisung zu treffen.

Switch-Anweisung

Eine switch-Anweisung wählt aus verschiedenen möglichen Anweisungsfolgen eine Folge zur Ausführung aus.

Lebensdauer

Die Lebensdauer einer Variablen legt fest, wie lange sie existiert, bevor sie zerstört wird.

lokale Variablen

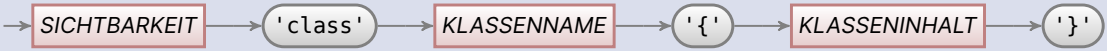
Eine lokale Variable ist eine Variable, die innerhalb einer Methode deklariert und benutzt wird. Sie ist nur innerhalb der Methode zugreifbar und ihre Lebensdauer entspricht der ihrer Methode.

Syntaxdiagramme I

Datenfelder, Instanzvariablen, Eigenschaften

Einfache Angabe der Sprachsyntax durch **Syntaxdiagramme**.

Verallgemeinert (Klassendefinition)



Am Beispiel (Klassendefinition)

Syntaxdiagramme II

Datenfelder, Instanzvariablen, Eigenschaften

Verallgemeinert

Am Beispiel

Syntaxdiagramme III

Datenfelder, Instanzvariablen, Eigenschaften

Verallgemeinert (Konstruktor)

Am Beispiel (Konstruktor)

Syntaxdiagramme IV

Datenfelder, Instanzvariablen, Eigenschaften

Verallgemeinert (*getter*-Methode), Datenfeld xyz

Am Beispiel (*getter*-Methode), Datenfeld `price`

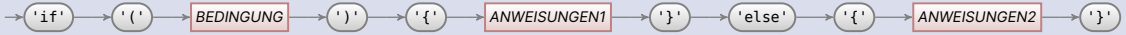
Verallgemeinert (*setter*-Methode, Datenfeld xyz)

Am Beispiel (setter Methode), Datenfeld `discount`

Syntaxdiagramme V

Datenfelder, Instanzvariablen, Eigenschaften

Verallgemeinert (Bedingte Anweisung)



Verallgemeinert (Switch-Anweisung)