Zusammenspiel von/Kommunikation zwischen Objekten

Buch, Kapitel 3

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

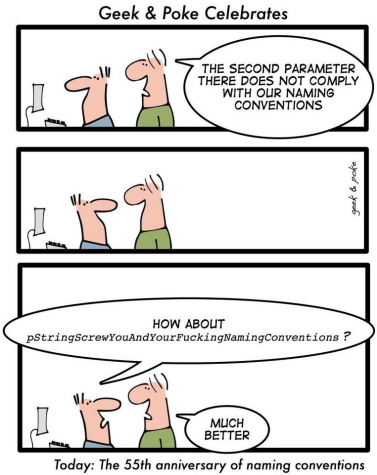
Studiengang Wirtschaftsinformatik

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

Vorlesungsinhalt

1. Lerninhalte
2. Beispielszenario Digitaluhr
3. Abstraktion und Modularisierung
4. Interaktion zwischen Objekten

   Namenskonflikte und Selbstreferenz
   Bedingungen: Logische Operatoren
5. Fehlersuche/-analyse



Lernziele

Nach Beenden dieser Lektion werden Sie

- Die Prinzipien Abstraktion und Modularisierung als Grundlage für einen Klassenentwurf verstehen und anwenden können.
- Externe und interne Methodenaufrufe programmieren können.
- Die logischen Operatoren von Java kennen und anwenden können.

Lerninhalte

- Operatoren für arithmetische und boolesche Ausdrücke
- Referenztypen
- Methodenaufrufe auf Objekten

Eine Digitaluhr

Das zu entwickelnde System soll

- die Möglichkeit eröffnen eine bestimmte Zeit einzustellen.
- die Zeit voranschreiten lassen.
- eine aufbereitete Ausgabe von Uhrzeiten zur Verfügung stellen.

Abstraktion und Modularisierung

Die Grundlage eines Softwareentwurfs besteht aus der Anwendung von Prinzipien, die sich bei der Entwicklung großer Programmsysteme bewährt haben.

- Abstraktion ist der Übergang von der Betrachtung einzelner individueller Objekte zu Klassen von Objekten mit gemeinsamen Eigenschaften. Dabei werden bestimmte Details ausgeblendet und andere als Gemeinsamkeiten hervorgehoben.
- Modularisierung ist das Zerlegen eines Problems in mehrere Teilprobleme. Sind die Teilprobleme gut gewählt, kann man sie isoliert voneinander lösen. Die Lösung des Gesamtproblems ergibt sich durch Zusammensetzung der Lösungen der Teilprobleme.

Quiz

Abstraktion und Modularisierung sind Schlüsselkonzepte des Software Engineering.
Welche Vorteile hat man, wenn man Software modularisiert?
Gibt es auch Nachteile?

Small is beautiful

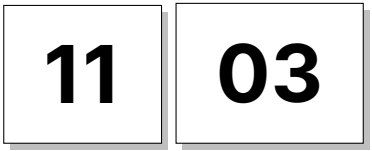
Es gibt verschiedene Möglichkeiten Modularisierung anzuwenden

1. Auf Klassenebene: Mehrere kleine Klassen sind besser als eine große. Die Unterscheidung zwischen klein und groß kann man anhand der „Lines of Code“, also der Zeilen ausführbaren Codes machen. Faustregel: Mehr als 50 Lines of Code ist groß, weniger klein.
2. Auf Methodenebene: Mehrere kurze Methoden sind besser als eine lange. Auch hier kann man die Unterscheidung anhand der „Method Lines of Codes“ machen. Mehr als 10 MLOC sind lang, weniger kurz. Häufig wird aber auch noch die „Schwierigkeit“ einer Methode mit betrachtet. Dabei werden die Alternativen und Schleifen mitgerechnet, die in einer Methode programmiert sind.

Modularisierung der Digitaluhr

eine vierstellige Anzeige?

Oder zwei zweistellige Anzeigen?



Modellierung eines zweistelligen Display

- Wir entwerfen eine Klasse `NumberDisplay`.
- Zwei Exemplarvariablen: `currentValue` und `limit`
- `currentValue` wird bis zum `limit` hochgezählt.
- Bei Überschreiten des `limit` wird wieder bei 0 angefangen.

Implementation

Klasse NumberDisplay

```
public class NumberDisplay {
  private int limit;
  private int currentValue;

  public NumberDisplay (int max) {
    limit = max;
    currentValue = 0;
  }
  ...
}
```

`increment` **Methode**

In der Klasse NumberDisplay

```
public void increment() {
  currentValue = currentValue + 1;
  if (currentValue == limit) {
    // Keep the value within the limit.
    currentValue = 0;
  }
}
```

increment Version 2

Die Modulo Rechnung kann verwendet werden um in unserem Beispiel eine einfachere Implementation zu erhalten:

increment ohne Alternative

```
public void increment() {
  currentValue = (currentValue + 1) % limit;
}
```

ClockDisplay

Die Klassendefinition von `ClockDisplay`

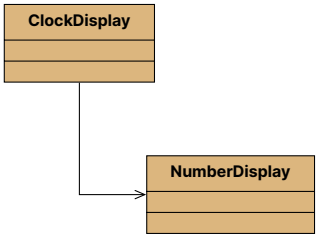
```
public class ClockDisplay {
  private NumberDisplay hours;
  private NumberDisplay minutes;

  //Konstruktor und Methoden weggelassen
}
```

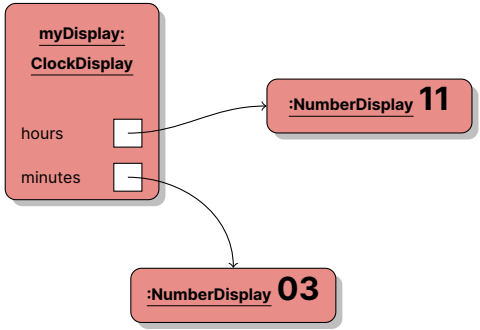
- Dieser Entwurf berücksichtigt Modularisierung, weil er ein größeres Problem in zwei kleinere zerlegt.
- Dieser Entwurf verwendet Abstraktion, weil er das Stunden- (24 Schritte) und das Minutenproblem (60 Schritte) in einer Klasse zusammenfasst.
- **Klassen definieren Typen**

Klassen als Typen

- Variablen in Java können verschiedene Typen haben: **`int`**, **`boolean`**, **`float`**, … .
- Diese Typen heißen auch primitive Datentypen, weil sie nicht aus anderen Datentypen zusammengesetzt sind.
- Variablen zu primitiven Datentypen repräsentieren Werte.
- Jede Klasse kann als Typ verwendet werden. Sie stehen dann als „zusammengesetzter“ Typ zur Verfügung. Beispiele hierfür sind `String`, `TicketMachine`, `NumberDisplay`, … .
- Variablen mit einer Klasse als Typ repräsentiert ein Objekt dieser Klasse.

Klassendiagramme und Objektdiagramme





Objekte mit new erzeugen

Objekte erzeugen Objekte

```
public class ClockDisplay {
  private NumberDisplay hours;
  private NumberDisplay minutes;

  public ClockDisplay() {
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    ...
  }
}
```

Quiz: Was ist die Ausgabe?

Wertzuweisung

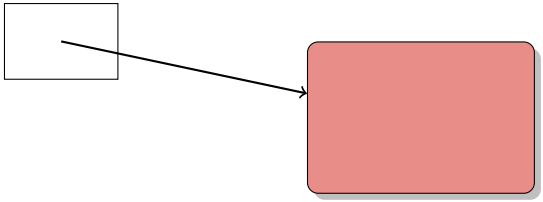
```
int a;
int b;
a = 32;
b = a;
a = a + 1;
System.out.println(b);
```

Objektzuweisung

```
Person a;
Person b;
a = new Person("Everett");
b = a;
a.changeName("Delmar");
System.out.println(b.getName());
```

Primitive Typen vs. Referenztypen

- `SomeClass obj;`
- Die Variable `obj` verweist (referenziert) auf ein Objekt der Klasse `SomeClass` (**Objektreferenz**).



- `int i;`
- Die Variable `i` hat den Wert 32.

Wirkung der Zuweisung b=a

Bei Referenztypen erhält die Variable b dieselbe Referenz wie die Variable a. Änderungen an dem Objekt wirken sich für beide Variablen aus, da sie auf das selbe Objekt verweisen.

Bei primitiven Typen erhält die Variable b denselben Wert wie die Variable a. Änderungen an dem Wert einer Variablen wirken sich nicht auf den Wert der anderen Variablen aus.

Interaktion zwischen Objekten

Durch die Modularisierung der Software ist es erforderlich, dass Objekte miteinander interagieren.

- Wenn zwei Objekte miteinander interagieren ruft eines eine Methode eines anderen Objekts auf.
- Die Interaktion ist unidirektional. Damit ergeben sich zwei Rollen, die des „Clientobjekts“ und die des „Serverobjekts“.
- Der Client kann den Server auffordern etwas zu tun.
- Der Client kann den Server nach Informationen fragen.
- Der Client muss den Server kennen.
- Es ist gute Programmierpraxis, dass der Server den Client nicht kennt und ihn deshalb weder beauftragen noch befragen kann. Insofern gibt es eine Hierarchiebeziehung zwischen Client und Server. Der Client ist dem Server übergeordnet.

Interaktion im Uhr-Szenario

- Zwei `NumberDisplay` Exemplare speichern Informationen für das `ClockDisplay` Exemplar.
- Das `ClockDisplay` Exemplar ist der Client.
- Die `NumberDisplay` Exemplare sind die Server.
- Die Methoden des Client-Objekts rufen die Methoden der Server Objekte auf.

Interne und externe Methodenaufrufe

- Ein **externer Methodenaufruf** ist ein Methodenaufruf eines anderen Objekts, unabhängig vom Typ.
- Ein Methodenaufruf auf einem Objekt, das derselben Klasse angehört ist auch ein externer Aufruf.
- Ein **interner Methodenaufruf** ist ein Aufruf einer Methode auf demselben Objekt.

Beispiel Methodenaufruf

Methode `timeTick` im Client

```
public void timeTick() {
  minutes.increment();
  if (minutes.getCurrentValue() == 0) {
    // it just rolled over!
    hours.increment();
  }
}
```

Externe Methodenaufrufe

- Stehen in Client Methoden.
- Generelle Form
  `serverObjektVariable.methodenname(parameterliste);`
- Beispiele:
  `hours.increment();`
  `minutes.increment();`

Interne Methodenaufrufe

- Keine Variable benötigt. Die Methode wird auf dem aktuellen Objekt ausgeführt.
- Interne Methoden haben häufig die Sichtbarkeit **`private`**.
- Beispiel:
  `updateDisplay();`
- Alternative: **`this`**`.updateDisplay();`

Das reservierte Wort `this`

- **`this`** steht für das aktuelle Objekt.
- Heißt ein Parameter einer Methode genau wie eine Exemplarvariable, so kann man durch Nutzung von **`this`** zwischen den Variablen unterscheiden.

- Die Namensgleichheit ist insbesondere bei Konstruktoren und setter-Methoden sinnvoll.

`this` zur Beseitigung eines Namenskonflikts

```
public class NumberDisplay {
private int limit;

public NumberDisplay(int limit) {
  this.limit = limit;
  currentValue = 0;
}

...
}
```

Logische Operatoren

Logische Operatoren erlauben es boolesche Bedingungen zu formulieren.

Operator	Bedeutung
&	Und; beide Argumente werden ausgewertet
|	Oder; beide Argumente werden ausgewertet
^	Exklusives oder; beide Argumente werden ausgewertet
!	Negation
&&	Und mit **Kurzschlussauswertung**
||	Oder mit **Kurzschlussauswertung**

Übung: Logische Operatoren

Welche der folgenden Ausdrücke liefern **`true`**?

- `! (4<5)`
- `! false`
- `2 > 2 || ((4 == 4) && (1 < 0))`
- `2 > 2 || (4 == 4) && (1 < 0)`
- `(34 != 33) && ! false`

Überlegen Sie sich die Antworten zunächst und notieren Sie diese. Nutzen Sie im Anschluss die Direkteingabe in BlueJ und überprüfen Sie ihre Einschätzungen.

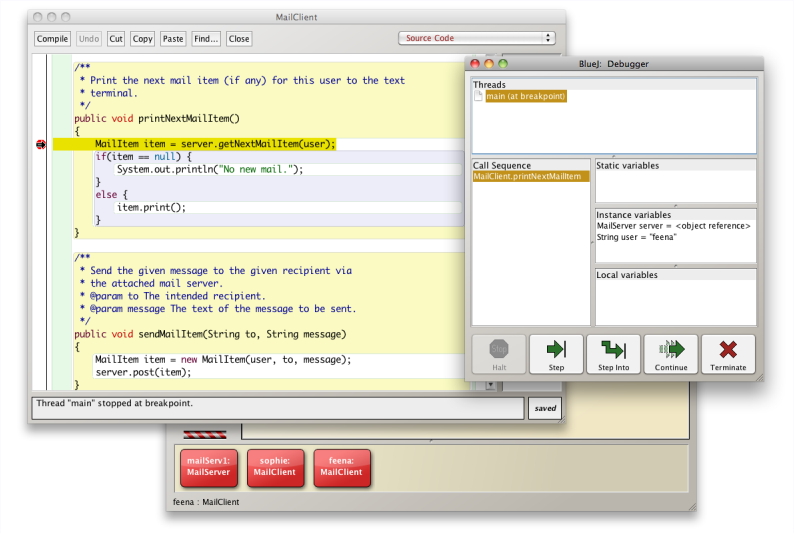
true and false

Schreiben Sie einen Ausdruck mit 2 booleschen Variablen a und b, der **`true`** liefert, wenn nur genau eine von beiden **`true`** ist, und der **`false`** liefert, wenn a und b beide **`false`** oder beide **`true`** sind (dies bezeichnet man auch als exklusives Oder)

Der Debugger

- Zum Nachvollziehen des Verhaltens eines Programmes, insbesondere wenn sich das Programm anders verhält, als man erwartet.
- Breakpoints (Unterbrechungspunkte) unterbrechen die Programmausführung
  - um Variableninhalte abzufragen.
  - um den Code schrittweise auszuführen.

Der Debugger in BlueJ



Fragen?

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

Konzepte: Zusammenfassung I

- **Abstraktion** Abstraktion ist die Fähigkeit Details von Bestandteilen zu ignorieren, um den Fokus der Betrachtung auf eine höhere Ebene lenken zu können.
- **Modularisierung** Modularisierung ist der Prozess der Zerlegung eines Ganzen in wohldefinierte Teile, die getrennt erstellt und untersucht werden können und die in wohldefinierter Weise interagieren.
- **Klassen definieren Typen** Ein Klassenname kann als Typname in einer Deklaration verwendet werden. Variable die eine Klasse als Typ haben können Objekte dieser Klasse halten/speichern.
- **Klassendiagramme** Ein Klassendiagramm zeigt die Klassen einer Anwendung und die Beziehung zwischen diesen Klassen. Es liefert Informationen über den Quelltext. Es präsentiert eine statische Sicht auf ein Programm.
- **Objektdiagramme** Ein Objektdiagramm zeigt die Objekte und ihre Beziehungen zu einem bestimmten Zeitpunkt während der Ausführung einer Anwendung. Es präsentiert eine dynamische Sicht auf ein Programm.
- **Objektreferenz** Variablen von Objekttypen speichern Referenzen auf Objekte.
- **externer Methodenaufruf** Methoden können Methoden von anderen Objekten über die Punkt-Notation aufrufen. Dies wird als externer Methodenaufruf bezeichnet.

Konzepte: Zusammenfassung II

- **interner Methodenaufruf** Methoden können andere Methoden ihrer Klasse als Teil ihrer Implementierung aufrufen. Dies wird als interner Methodenaufruf bezeichnet.
- **Debugger** Ein Debugger ist ein Softwarewerkzeug, mit dessen Hilfe die Ausführung eines Programms untersucht werden kann. Es kann benutzt werden, um Fehler (Bugs) zu finden.

Abstraktion

Abstraktion ist die Fähigkeit Details von Bestandteilen zu ignorieren, um den Fokus der Betrachtung auf eine höhere Ebene lenken zu können.

Modularisierung

Modularisierung ist der Prozess der Zerlegung eines Ganzen in wohldefinierte Teile, die getrennt erstellt und untersucht werden können und die in wohldefinierter Weise interagieren.

Klassen definieren Typen

Ein Klassenname kann als Typname in einer Deklaration verwendet werden. Variable die eine Klasse als Typ haben können Objekte dieser Klasse halten/speichern.

Klassendiagramme

Ein Klassendiagramm zeigt die Klassen einer Anwendung und die Beziehung zwischen diesen Klassen. Es liefert Informationen über den Quelltext. Es präsentiert eine statische Sicht auf ein Programm.

Objektdiagramme

Ein Objektdiagramm zeigt die Objekte und ihre Beziehungen zu einem bestimmten Zeitpunkt während der Ausführung einer Anwendung. Es präsentiert eine dynamische Sicht auf ein Programm.

Objektreferenz

Variablen von Objekttypen speichern Referenzen auf Objekte.

externer Methodenaufruf

Methoden können Methoden von anderen Objekten über die Punkt-Notation aufrufen. Dies wird als externer Methodenaufruf bezeichnet.

interner Methodenaufruf

Methoden können andere Methoden ihrer Klasse als Teil ihrer Implementierung aufrufen. Dies wird als interner Methodenaufruf bezeichnet.

Debugger

Ein Debugger ist ein Softwarewerkzeug, mit dessen Hilfe die Ausführung eines Programms untersucht werden kann. Es kann benutzt werden, um Fehler (Bugs) zu finden.