Tutorium zur Vorlesung Programmieren

4. Tutorium

Joshua Gleitze

21. November 2016

IPD Koziolek

Korrektur des Übungsblatts

Praktomat

Attestation: Übungsblatt 1 - Teil E

by Joshua Gleitze for

Comment

Hier steht ein Kommentar, der die gesamte Abgabe betrifft

Ratings

Final grade: 9.0

Checker results

Copy File : passed

▸ **Java - Compiler** : passed

Copy File : passed

▸ **Pakete, Klassen Checks** : passed

Annotated Files

Download Solution

Months.java | Time.java | TimePoint.java | TimeZone.java | ZoneTime.java | Date.java* | Weekdays.java*

```
1   public class Date {
2 +     //! Hier steht ein Kommentar, der sich jeweils auf die Zeile(n) darüber bezieht
```

Formulierungen auf dem Übungsblatt

Aussagen auf dem Übungsblatt sind bindend!

„Programmcode muss in englischer Sprache verfasst sein.“

Enum

Enums müssen nicht innerhalb einer Klasse stehen
(Das ist auch nur selten sinnvoll)

Season.java

```java
enum Season {
    SPRING,
    SUMMER,
    AUTUMN,
    WINTER
}
```

Jacket.java

```java
class Jacket {
    String name;
    double price;
    Season forSeason;
}
```

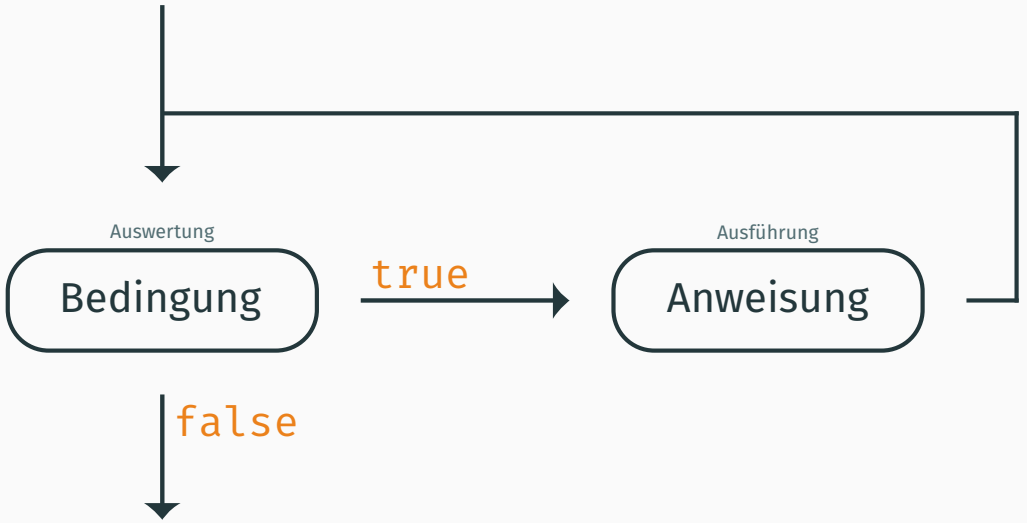
Shop.java (Auszug)

```java
Jacket favouriteJacket = new Jacket();
favouriteJacket.name = "Strellson MATT";
favouriteJacket.price = 299.99;
favouriteJacket.forSeason = Season.WINTER;
```

Kontrollstrukturen

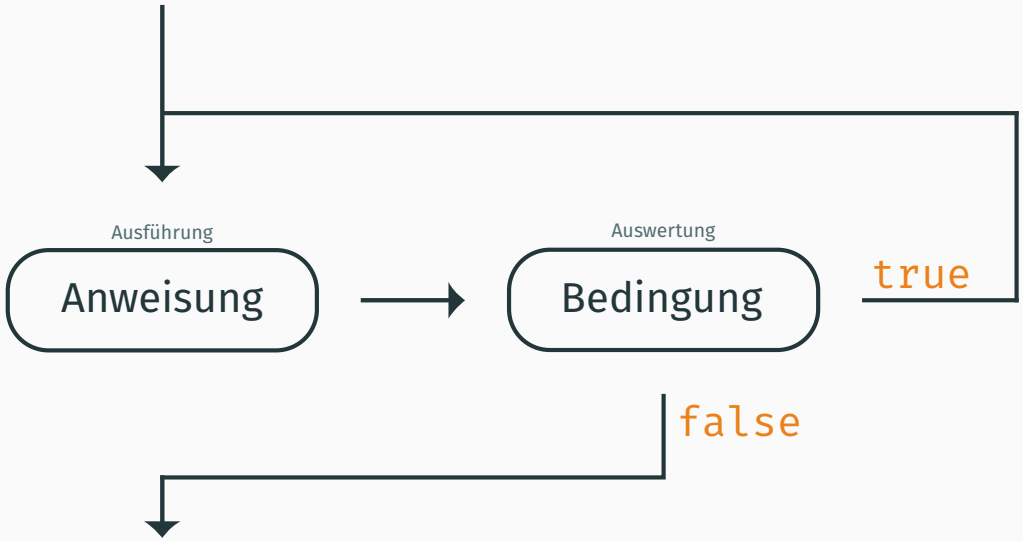
While-Schleife

```
1  while (Bedingung)
2      Anweisung
```



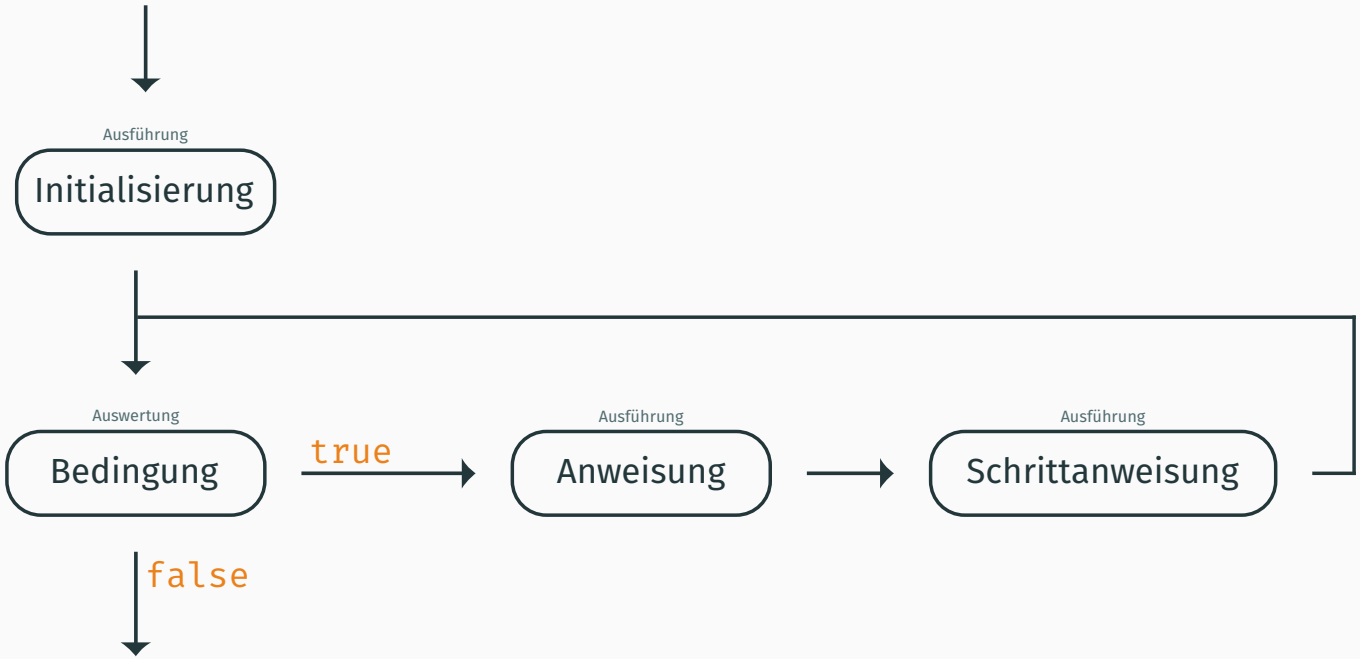
Do-While-Schleife

```
1  do
2      Anweisung
3  while (Bedingung);
```



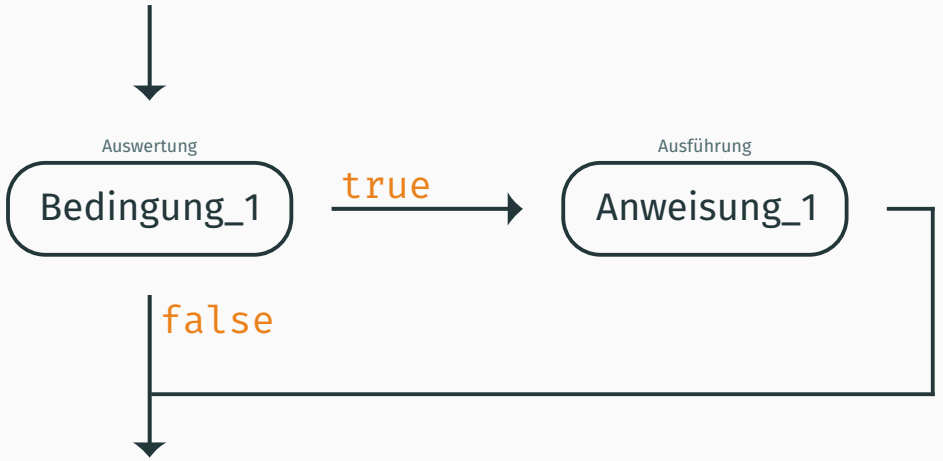
For-Schleife

```
1  for (Initialisierung; Bedingung; Schrittanweisung)
2      Anweisung
```



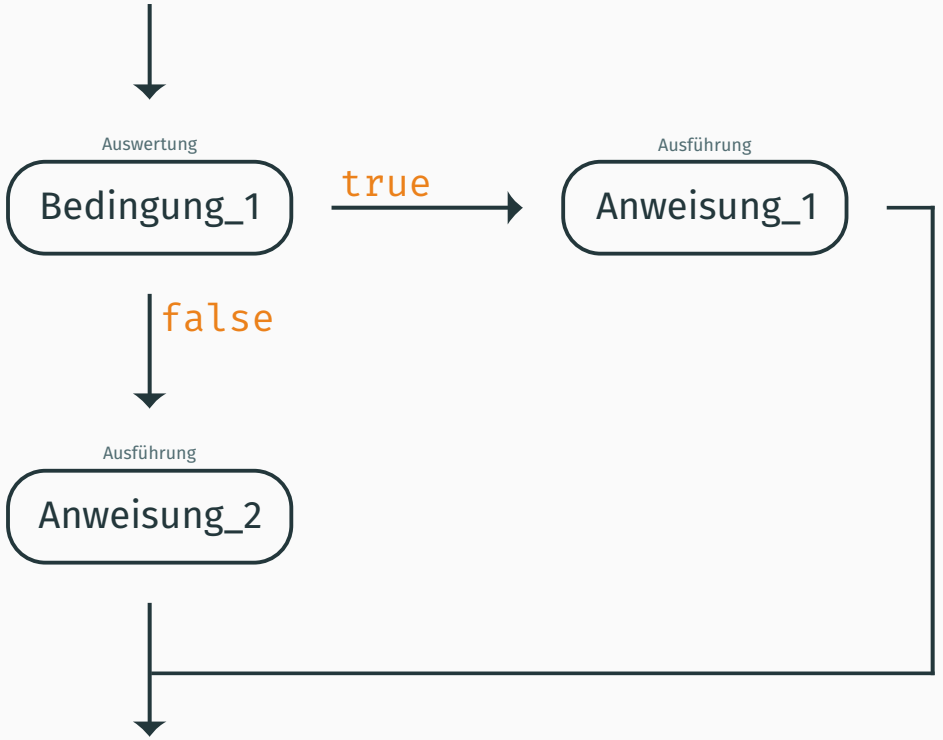
If-Bedingung

```
1  if (Bedingung_1)
2      Anweisung_1
```



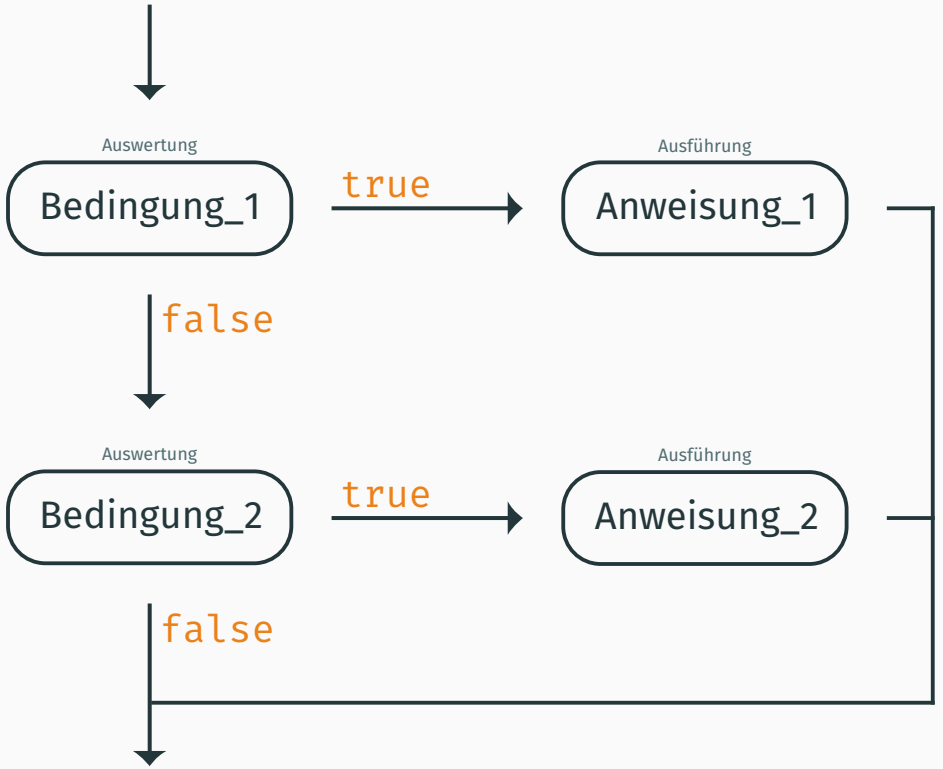
If-Bedingung

```
1  if (Bedingung_1)
2      Anweisung_1
3  else
4      Anweisung_2
```



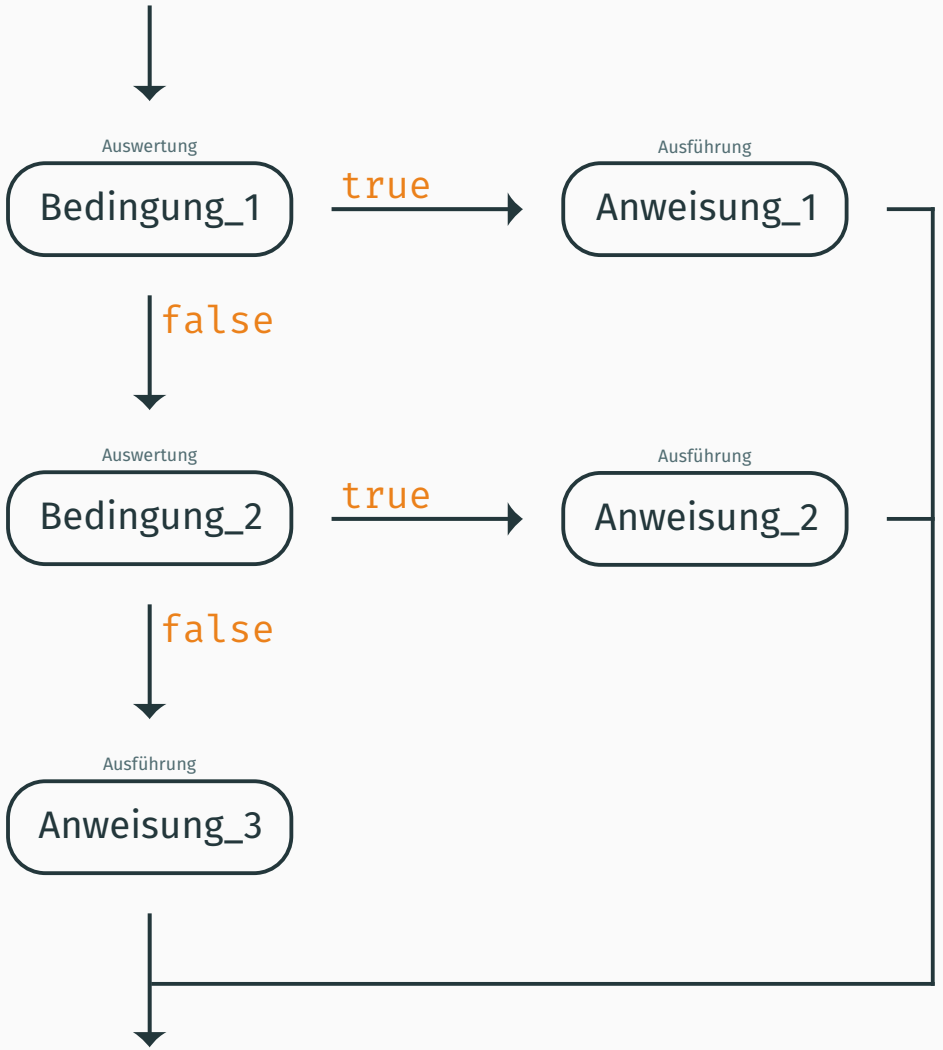
If-Bedingung

```
1  if (Bedingung_1)
2      Anweisung_1
3  else if (Bedingung_2)
4      Anweisung_2
```



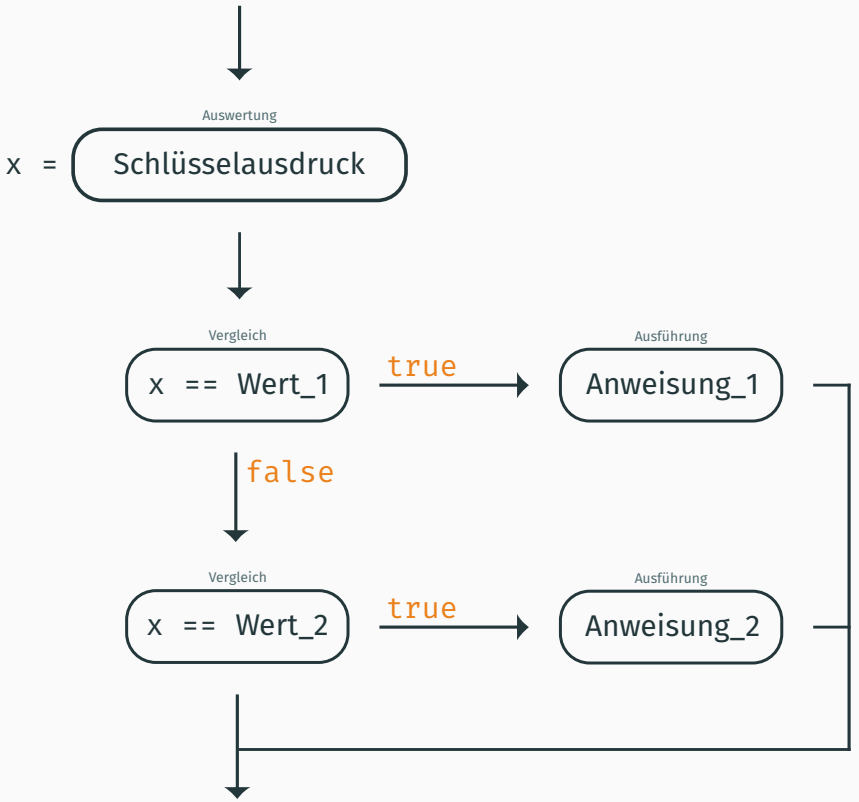
If-Bedingung

```
1  if (Bedingung_1)
2      Anweisung_1
3  else if (Bedingung_2)
4      Anweisung_2
5  else
6      Anweisung_3
```



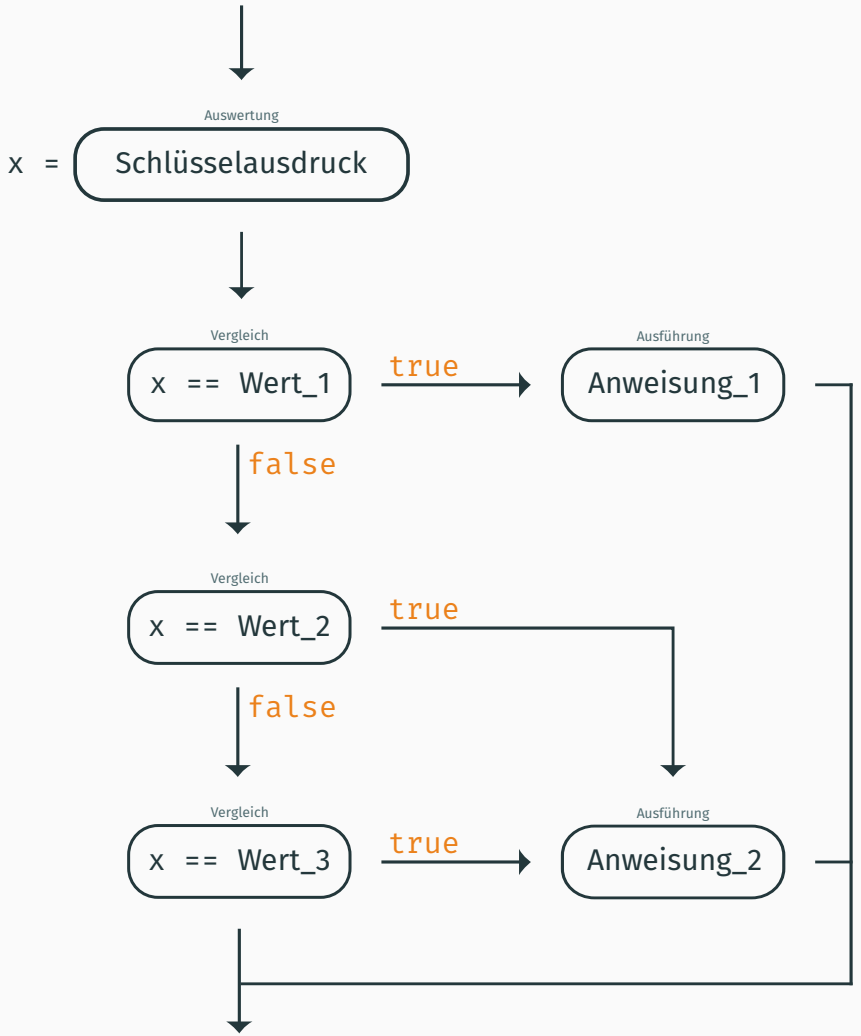
Switch

```
1  switch (Schlüsselausdruck)
   ↪  {
2      case Wert_1:
3          Anweisung_1
4          break;
5      case Wert_2:
6          Anweisung_2
7          break;
8  }
```



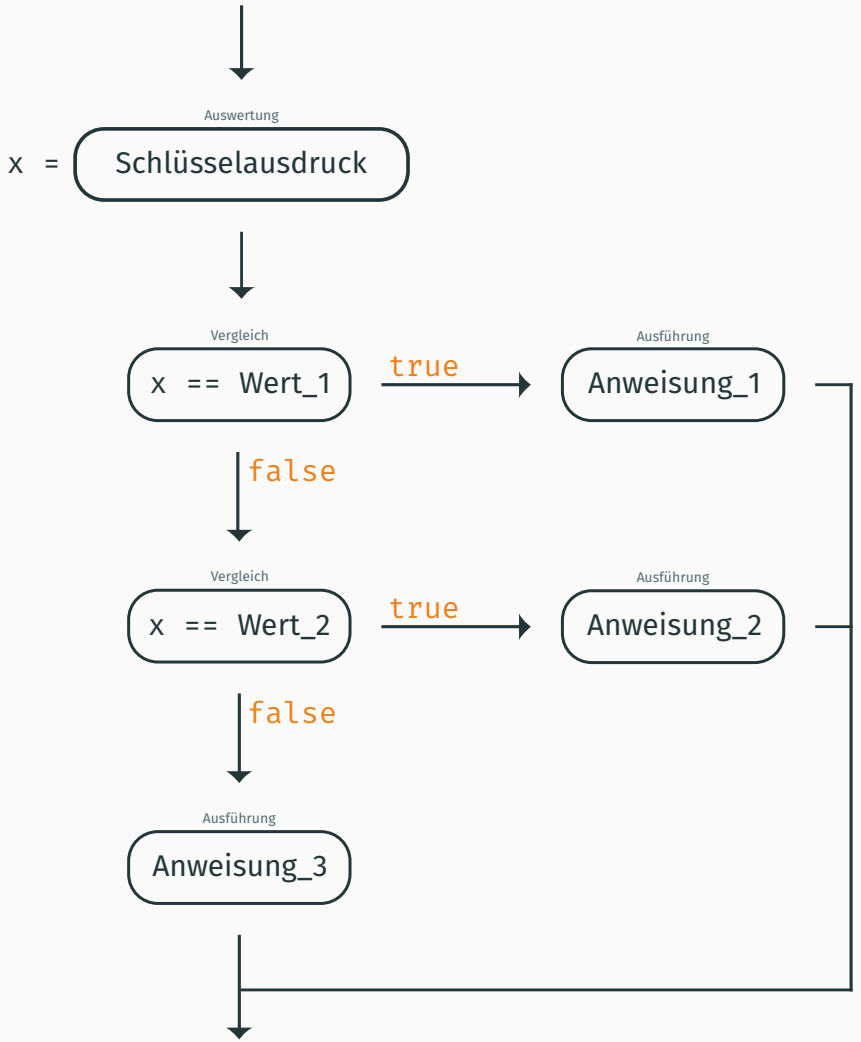
Switch

```
1  switch (Schlüsselausdruck)
   ↪  {
2      case Wert_1:
3          Anweisung_1
4          break;
5      case Wert_2:
6      case Wert_3:
7          Anweisung_2
8          break;
9  }
```



Switch

```
1  switch (Schlüsselausdruck)
   ↪  {
2      case Wert_1:
3          Anweisung_1
4          break;
5      case Wert_2:
6          Anweisung_2
7          break;
8      default:
9          Anweisung_3
10 }
```



Switch

```
1  public class Switch {
2      public void main(String[] args) {
3          switch(args[0]) {
4              case "hi":
5                  System.out.println("Hallo!");
6              case "bye":
7                  System.out.println("Tschüss!");
8          }
9      }
10 }
```

```
java Switch hi
```

Switch

```
1  public class Switch {
2      public void main(String[] args) {
3          switch(args[0]) {
4              case "hi":
5                  System.out.println("Hallo!");
6              case "bye":
7                  System.out.println("Tschüss!");
8          }
9      }
10 }
```

```
java Switch hi
Hallo!
Tschüss!
```

In Switch `break` nicht vergessen!

Objekte und Methoden

NEW

`new` erstellt ein neues Objekt einer bestimmten Klasse

THIS

`this` referenziert die eigene Instanz

Konstruktoren

- Jede Klasse hat einen default-Konstruktor
- Implementiert man einen Kostruktor, „verschwindet“ der default-Kostruktor
- Syntax:

```
Klassenname(Typ1 parameter1, Typ2 parameter2) {
    Anweisungen
}
```

- Aufruf: `new Klassenname(parameter1, parameter2);`

Konstruktoren

File.java

```java
class File {
    String path;

    File(String path) {
        this.path = path;
    }
}
```

Konstruktoren

File.java

```java
class File {
    String path;

    File(String path) {
        this.path = path;
    }
}
```

Neues `File`:

```java
File i3config = new File("~/.config/i3/config");
```

Konstruktoren

File.java

```
1  class File {
2      String path;
3
4      File(String path) {
5          this.path = path;
6      }
7  }
```

Nicht möglich:

```
1  File emptyFile = new File();
```

Konstruktoren

File.java

```java
class File {
    String path;

    File() {}

    File(String path) {
        this.path = path;
    }
}
```

Jetzt möglich:

```java
File emptyFile = new File();
```

FINAL

`final`

- signalisiert: keine Änderung nach Initialisierung
- für Attribute und Variablen
- `final` Attribute müssen im Konstruktor oder bei Deklaration initialisiert werden

FINAL

Fehler beim Kompilieren:

```
1  class Account {
2      final long number;
3  }
```

FINAL

Möglich, aber nicht sinnvoll:

```
1  class Account {
2      final long number = 0;
3  }
```

FINAL

Fehler beim Kompilieren:

```
1  class Account {
2      final long number;
3
4      Account() {}
5  }
```

FINAL

Möglich, aber nicht sinnvoll:

```
1  class Account {
2      final long number;
3
4      Account() {
5          this.number = 0
6      }
7  }
```

FINAL

Sinnvoll:

```
1  class Account {
2      final long number;
3
4      Account(long number) {
5          this.number = number;
6      }
7  }
```

Methoden

```
1  Typ methodenName(Typ1 parameter1, Typ2 Parameter2) {
2      Anweisungen
3      return Ergebnis;
4  }
```

Aufruf: `methodenName(parameter1, parameter2);`

Methoden

```
1  class Account {
2      final long number;
3      long balance;
4
5      ...
6
7
8      boolean transferTo(Account foreign, long amount) {
9          if (this.balance < amount) {
10             return false;
11         }
12         this.balance -= amount;
13         foreign.balance += amount;
14         return true;
15     }
16 }
```

Methoden

Account.java

```java
class Account {
    ...

    boolean transferTo(Account foreign, long amount) {
        if (this.balance < amount) {
            return false;
        }
        this.balance -= amount;
        foreign.balance += amount;
        return true;
    }
}
```

Bank.java (Auszug)

```java
System.out.println(kitAccount.balance);         // 25000
System.out.println(joshuasAccount.balance);     // 23

boolean success = kitAccount.transferTo(joshuasAccount, 3000);

System.out.println(kitAccount.balance);         // 22000
System.out.println(joshuasAccount.balance)      // 3023
System.out.println(success);                    // true
```

Methoden

Account.java

```java
class Account {
    ...

    boolean transferTo(Account foreign, long amount) {
        if (this.balance < amount) {
            return false;
        }
        this.balance -= amount;
        foreign.balance += amount;
        return true;
    }
}
```

Bank.java (Auszug)

```java
System.out.println(kitAccount.balance);         // 22000
System.out.println(joshuasAccount.balance);     // 3023

boolean success = joshuasAccount.transferTo(kitAccount, 3030);

System.out.println(kitAccount.balance);         // 22000
System.out.println(joshuasAccount.balance)      // 3023
System.out.println(success);                    // false
```

NULL

- `null` ist ein Platzhalter für ein nicht vorhandenes Objekt
- `null` hat jeden Typ
- Ein Aufruf auf `null` führt immer zu einer `NullPointerException`

VOID

`void` signalisiert „keine Rückgabe“

$\neq$ `null`!

Getter und Setter

Getter und Setter sind *Konvention*

Direkter Zugriff auf Attribute nur für `static final`

Getter und Setter

```
1  class Account {
2      final long number;
3      long balance;
4
5      ...
6
7      long getNumber() {
8          return this.number;
9      }
10
11     void setBalance(long balance) {
12         this.balance = balance;
13     }
14
15     int getBalance() {
16         return this.balance;
17     }
18 }
```

Begriffsklärung

Objektorientierte Modellierung

Klasse – Objekt – Methode – Attribut – Variable

Klasse – Objekt – Methode – Attribut – Variable

Account.java

```java
class Account {
    final long number;
    long balance;

    ...

    long effectiveInterest() {
        long interest = this.balance * this.getInterestRate();
        long fees = this.getStatementFees()
            + this.getAccountFees()
            + this.getWithdrawalFees();
        return interest - fees;
    }
}
```

Klasse – Objekt – Methode – Attribut – Variable

Bank.java (Auszug)

```java
Account joshuasAccount = new Account(5468135467);
Account kitAccount = new Account(4489984654);
```

Klasse – Objekt – Methode – Attribut – Variable

Account.java (Auszug)

```java
long effectiveInterest() {
        long interest = this.balance * this.getInterestRate();
        long fees = this.getStatementFees()
            + this.getAccountFees()
            + this.getWithdrawalFees();
        return interest - fees;
    }
```

Bank.java (Auszug)

```java
long small = joshuasAccount.getEffectiveInterest();
```

Klasse – Objekt – Methode – Attribut – Variable

Account.java (Auszug)

```java
class Account {
    final long number;
    long balance;
```

Bank.java (Auszug)

```java
joshuasAccount.balance = 20000;
```

Klasse – Objekt – Methode – Attribut – Variable

Account.java (Auszug)

```java
long effectiveInterest() {
    long interest = this.balance
            * this.getInterestRate();
    long fees = this.getStatementFees()
        + this.getAccountFees()
        + this.getWithdrawalFees();
    return interest - fees;
```

Bestandteile eines Programms

Programm – Klasse – `main`-Methode

Bestandteile eines Programms

Programm – Klasse – `main`-Methode

Account.java

```
1 class Account {
2     ...
```

Bank.java

```
1 class Bank {
2     ...
```

Currency.java

```
1 enum Currency {
2     ...
```

MainWindow.java

```
1 class MainWindow {
2     ...
```

Customer.java

```
1 class Customer {
2     ...
```

Transfer.java

```
1 class Transfer {
2     ...
```

...

Bestandteile eines Programms

Programm – Klasse – `main`-Methode

Account.java

```
1  class Account {
2      final long number;
3      long balance;
4
5      ...
6
7      long effectiveInterest() {
8          long interest = this.balance * this.getInterestRate();
9          long fees = this.getStatementFees()
10             + this.getAccountFees()
11             + this.getWithdrawalFees();
12         return interest - fees;
13     }
14 }
```

Bestandteile eines Programms

Programm – Klasse – `main`-Methode

Eine pro Programm

Bank.java

```
1  class Bank {
2      public static void main (String[] args) {
3          MainWindow gui = new MainWindow();
4          gui.show();
5          ...
```

Stil

Aussagekräftige Namen

Vom Namen sollte man auf die Funktion schließen können.

Kommentare

Klassen und Attribute

- Die Funktion und was modelliert wird sollte ohne Lektüre des Quelltexts klar werden

Kommentare

Methoden

- Das Verhalten sollte ohne Lektüre des Quelltexts klar werden
- Zu jeder Methode gehört mindestens eine vollständige Spezifikation der Ein- und Ausgabe

Javadoc

- Kommentare beginnen mit `/**`
- Für Klassen, Enums, Methoden, Attribute, Pakete
- kann in HTML-Dokumentation gerendert werden
- 1. Satz einer Beschreibung erscheint in der Übersicht

Javadoc: HTML-Rendering

de.uka.ipd.sdq.beagle.core

Class Blackboard

java.lang.Object
    de.uka.ipd.sdq.beagle.core.Blackboard

All Implemented Interfaces:

`Serializable`

```
public class Blackboard
extends Object
implements Serializable
```

Central and only storage of all knowledge gained by Beagle. Implements, together with `AnalysisController`, the Blackboard pattern from POSA I. The Blackboard's vocabularies are: `ResourceDemandingInternalAction`, `SeffBranch`, `SeffLoop`, `ResourceDemandMeasurementResult`, `BranchDecisionMeasurementResult` , `LoopRepetitionCountMeasurementResult` and `EvaluableExpression`. It further allows classes to store custom data.

The Blackboard is typically not accessed directly by its using classes, but through *blackboard views* (recognisable by having the `BlackboardView` suffix). These are surrogates for the blackboard. They don't modify its contents but only restrict access to it.

See Also:

`AnalysisController`, `Serialized Form`

Javadoc: HTML-Rendering

Constructor Summary

Constructors

Constructor and Description
`Blackboard(Set<ResourceDemandingInternalAction> rdias, Set<SeffBranch> branches, Set<SeffLoop> loops, Set<ExternalCallParameter> externalCalls, EvaluableExpressionFitnessFunction fitnessFunction, ProjectInformation projectInformation)` Creates a new blackboard that can be used to analyse the given elements.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`addMeasurementResultFor(ExternalCallParameter parameter, ParameterChangeMeasurementResult results)` Adds a measurement result for the provided loop.
void	`addMeasurementResultFor(ResourceDemandingInternalAction rdia, ResourceDemandMeasurementResult results)` Adds a measurement result for the provided rdia.
void	`addMeasurementResultFor(SeffBranch branch, BranchDecisionMeasurementResult results)` Adds a measurement result for the provided branch.
void	`addMeasurementResultFor(SeffLoop loop, LoopRepetitionCountMeasurementResult results)` Adds a measurement result for the provided loop.
void	`addProposedExpressionFor(MeasurableSeffElement element, EvaluableExpression expression)` Adds expression as a proposal.

Javadoc-Tags

Für Methoden

```
/**
 * Beschreibung der Methode
 *
 * @param Parametername1 Beschreibung1
 * @return Beschreibung der Rückgabe
 */
```

Javadoc-Tags

Für Klassen

```
/**
 * Beschreibung der Klasse
 *
 * @author 1. Autor
 * @author 2. Autor
 * @version Version der Klasse
 */
```

Javadoc-Tags

Im Beschreibungstext

- `{@code Text}` für ein Codeschnipsel
  - Zum Beispiel Parameternamen
- `{@link Name}` für ein Link zu anderem Element
  - Syntax: `[Paketpfad.][Klasse]#attributOderMethode`
  - Zum Beispiel: `java.util.List#add`, `#getName`, `java.util.Set`, `Book#name`
- Bestimmtes HTML-Markup
  - Zum Beispiel `<p>`, `<em>`, `<strong>`, Tabellen

Javadoc: Beispiel

```java
/**
 * Central and only storage of all knowledge gained by Beagle. Implements, together with
 * {@link AnalysisController}, the Blackboard pattern from POSA I. The Blackboard's
 * vocabularies are: {@link ResourceDemandingInternalAction}, {@link SeffBranch},
 * {@link SeffLoop}, {@link ResourceDemandMeasurementResult},
 * {@link BranchDecisionMeasurementResult} , {@link LoopRepetitionCountMeasurementResult}
 * and {@link EvaluableExpression}. It further allows classes to store custom data.
 *
 * <p>The Blackboard is typically not accessed directly by its using classes, but through
 * <em>blackboard views</em> (recognisable by having the {@code BlackboardView} suffix).
 * These are surrogates for the blackboard. They don't modify its contents but only
 * restrict access to it.
 *
 * @author Christoph Michelbach
 * @author Joshua Gleitze
 * @author Roman Langrehr
 * @author Michael Vogt
 * @see AnalysisController
 */
public class Blackboard implements Serializable {

    /**
     * Serialisation version UID, see {@link java.io.Serializable}.
     */
    private static final long serialVersionUID = 6382577321150787599L;

    ...
```

Javadoc: Beispiel

```java
/**
 * Creates a new blackboard that can be used to analyse the given elements.
 *
 * @param rdias All resource demanding internal action to be known to analysers.
 * @param branches All SEFF branches to be known to analysers.
 * @param loops All SEFF loops to be known to analysers.
 * @param externalCalls All external call parameter to be known to analysers.
 * @param fitnessFunction The function to get better evaluable expression results.
 * @param projectInformation Information about the project belonging to this
 *            blackboard.
 */
public Blackboard(final Set<ResourceDemandingInternalAction> rdias,
    final Set<SeffBranch> branches,final Set<SeffLoop> loops,
    final Set<ExternalCallParameter> externalCalls,
    final EvaluableExpressionFitnessFunction fitnessFunction,
    final ProjectInformation projectInformation) {
    ...
}

/**
 * Returns the final expression set for {@code element}. The return value of this
 * method may change if
 * {@link #setFinalExpressionFor(MeasurableSeffElement, EvaluableExpression)} is
 * called with the same {@code element} between calls to
 * {@code addProposedExpressionFor} with this element as parameter.
 *
 * @param element A SEFF element. Must not be {@code null}.
 * @return The expression momentarily marked to be the final for {@code element}.
 *         {@code null} if no expression has been marked yet.
 */
public EvaluableExpression getFinalExpressionFor(final MeasurableSeffElement element) {
    ...
}
```

Javadoc

- Noch keine Pflicht auf dem Übungsblatt
- Struktiert aber Kommentare
- Lang, aber sehr lesenswert: „Oracle: How to Write Doc Comments for the Javadoc Tool“

Kommentare

Code

- Komplizierte / nicht naheliegende Schritte kommentieren
- Viele Kommentare nötig? $\Longrightarrow$ Code-Smell

Übung

Gibt es Fragen zum Übungsblatt?

Übungsaufgaben

https://judge.joshuagleitze.de

- Aufgaben sind im DOMJudge hinterlegt
- Aufgaben nehmen in der Schwierigkeit zu
- Prüft eure Abgabe bevor ihr sie hochladet