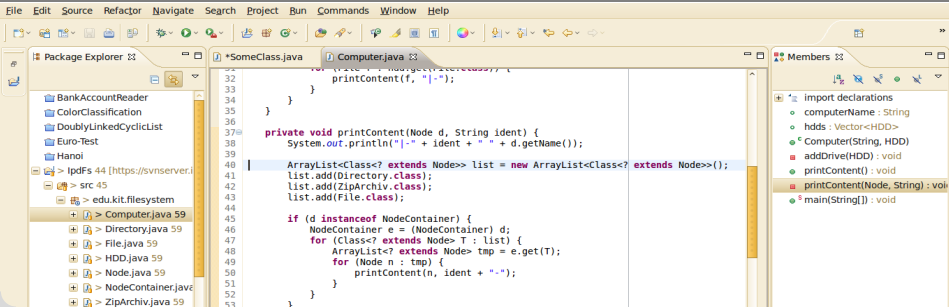


Programmieren-Tutorium Nr. 10 bei Martin Thoma

Pakete

Martin Thoma | 26. November 2012

FAKULTÄT FÜR INFORMATIK



The screenshot shows an IDE with the following components:

- Package Explorer (Left):** Displays a project structure with packages like `edu.kit.filesystem` and classes like `Computer.java`, `Directory.java`, `File.java`, `HDD.java`, `Node.java`, `NodeContainer.java`, and `ZipArchiv.java`.
- Editor (Center):** Shows the source code of `Computer.java`. The code includes a `printContent` method and a `main` method. The line `ArrayList<Class? extends Node> list = new ArrayList<Class? extends Node>();` is highlighted.
- Members (Right):** Displays the members of the `Computer` class, including `computerName`, `hdds`, `addDrive`, `printContent`, and `main`.

```
32         printContent(f, "|-");
33     }
34 }
35
36
37 private void printContent(Node d, String ident) {
38     System.out.println(f"|- " + ident + " " + d.getName());
39 }
40
41 ArrayList<Class? extends Node> list = new ArrayList<Class? extends Node>();
42 list.add(Directory.class);
43 list.add(ZipArchiv.class);
44 list.add(File.class);
45
46 if (d instanceof NodeContainer) {
47     NodeContainer e = (NodeContainer) d;
48     for (Class<? extends Node> T : list) {
49         ArrayList<? extends Node> tmp = e.get(T);
50         for (Node n : tmp) {
51             printContent(n, ident + "-");
52         }
53     }
54 }
```

- 1 Einleitung
- 2 ÜB 2
- 3 Vorlesungsergänzungen
- 4 ÜB 3
- 5 Praxis
- 6 Abspann

```
_____ String.java _____  
public class String {  
  
    java.lang.String content;  
  
    public String(java.lang.String content) {  
        this.content = content;  
    }  
}
```

```
_____ World.java _____  
public class World {  
    public static void main(String[] args) {  
        String a = new String("Lorem ipsum");  
        System.out.println(a);  
    }  
}
```

- Gibt es einen Compiler-Fehler?
- Gibt es einen Laufzeit-Fehler?
- Gibt es eine Ausgabe? Welche?

```
Exception in thread "main" java.lang.NoSuchMethodError: main  
Es gibt nun keine  
public static void main(java.lang.String[] args) {
```

Lehre

Keine Java-internen Typen umschreiben.

Musterlösung

Inoffizielle Musterlösung von Simon und mir ist unter
<http://goo.gl/BfA6i> erhältlich.
Bitte dort anschauen.

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
 - „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
 - Große Probleme (→ lange Methoden) aufsplitten
-
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
 - Kein „TODO“ in Abgaben
 - Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
 - Laufvariablen müssen nicht immer `i` und `j` heißen
 - Schaut euch die hochgeladenen Dateien im Praktomat an
 - Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)` , sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Stil

- Niemals `if (variable == true)`, sondern `if (variable)`
- „Dead Code“ - also Code der niemals erreicht wird - wird bestraft
- Große Probleme (→ lange Methoden) aufsplitten
- Die bereitgestellten Code-Vorlagen sind keine Musterlösungen!
- Kein „TODO“ in Abgaben
- Aufgabe B1 (Histogramm) war fehlerhaft: Natürlich muss der Kontrast nur „nicht-negativ“ sein und nicht „positiv“
- Laufvariablen müssen nicht immer `i` und `j` heißen
- Schaut euch die hochgeladenen Dateien im Praktomat an
- Redundanter Code ist schlecht ⇒ Besser: Neue Methode anlegen

Genau lesen

- **A.1.1 Konstruktor von Bike:** „Modifizieren Sie den Konstruktor der Klasse Bike, indem Sie aus der Signatur das Argument, nach dem bisher der Preis gesetzt wird, entfernen.“
 - **A.1.2 Setter für Gears** „Schreiben Sie den Konstruktor so um, dass er die Methode setSprockets benutzt, um einen konsistenten Anfangszustand zu garantieren.“
 - **A.2.1 Tribonacci-Folge:** „Geben Sie **nur** die siebenunddreißigste Tribonacci-Zahl auf der Konsole aus.“
- ⇒ Es werden automatische Tests durchgeführt. Stimmt die Ausgabe nicht exakt - also jedes einzelne (Leer)zeichen -, schlägt der Test fehl.

Genau lesen

- **A.1.1 Konstruktor von Bike:** „Modifizieren Sie den Konstruktor der Klasse Bike, indem Sie aus der Signatur das Argument, nach dem bisher der Preis gesetzt wird, entfernen.“
- **A.1.2 Setter für Gears** „Schreiben Sie den Konstruktor so um, dass er die Methode setSprockets benutzt, um einen konsistenten Anfangszustand zu garantieren.“
- **A.2.1 Tribonacci-Folge:** „Geben Sie **nur** die siebenunddreißigste Tribonacci-Zahl auf der Konsole aus.“

⇒ Es werden automatische Tests durchgeführt. Stimmt die Ausgabe nicht exakt - also jedes einzelne (Leer)zeichen -, schlägt der Test fehl.

Genau lesen

- **A.1.1 Konstruktor von Bike:** „Modifizieren Sie den Konstruktor der Klasse Bike, indem Sie aus der Signatur das Argument, nach dem bisher der Preis gesetzt wird, entfernen.“
- **A.1.2 Setter für Gears** „Schreiben Sie den Konstruktor so um, dass er die Methode setSprockets benutzt, um einen konsistenten Anfangszustand zu garantieren.“
- **A.2.1 Tribonacci-Folge:** „Geben Sie **nur** die siebenunddreißigste Tribonacci-Zahl auf der Konsole aus.“

⇒ Es werden automatische Tests durchgeführt. Stimmt die Ausgabe nicht exakt - also jedes einzelne (Leer)zeichen -, schlägt der Test fehl.

Genau lesen

- **A.1.1 Konstruktor von Bike:** „Modifizieren Sie den Konstruktor der Klasse Bike, indem Sie aus der Signatur das Argument, nach dem bisher der Preis gesetzt wird, entfernen.“
 - **A.1.2 Setter für Gears** „Schreiben Sie den Konstruktor so um, dass er die Methode setSprockets benutzt, um einen konsistenten Anfangszustand zu garantieren.“
 - **A.2.1 Tribonacci-Folge:** „Geben Sie **nur** die siebenunddreißigste Tribonacci-Zahl auf der Konsole aus.“
- ⇒ Es werden automatische Tests durchgeführt. Stimmt die Ausgabe nicht exakt - also jedes einzelne (Leer)zeichen -, schlägt der Test fehl.

```
System.arraycopy(warehouse, 0, newStock, 0, warehouse.length);
```

Gears.java

```
/**
 * Sets the sprocket numbers.
 * Uses default-values if consistency criteria are not met.
 * @param sprockets
 * @param rearSprockets
 */
void setSprockets(int sprockets, int rearSprockets) {
    this.frontSprockets = sprockets;
    this.rearSprockets = rearSprockets;

    if (!(this.frontSprockets >= 1)) { // A.1
        this.frontSprockets = 1;
    } else if (!(this.frontSprockets < 4)) { // A.2
        this.frontSprockets = 3;
    }

    // B.1, B.2
    if (this.rearSprockets < 1 || this.rearSprockets > 9) {
        this.rearSprockets = this.frontSprockets * 3;
    }

    if (this.rearSprockets < this.frontSprockets) { // C.1
        this.rearSprockets = this.frontSprockets;
    } else if (this.rearSprockets > 3 * this.frontSprockets) { // C.2
        this.rearSprockets = 3 * this.frontSprockets;
    }
}
```

Einheiten **immer** angeben, da ...

- nie klar ist, welche Einheit gemeint ist
- es ein richtig ärgerlicher Fehler ist
- fehlende Einheiten viel Geld kosten können (→ [Video: NASA Measuring Failure](#) - Mars Orbiter)

Einheiten **immer** angeben, da ...

- nie klar ist, welche Einheit gemeint ist
- es ein richtig ärgerlicher Fehler ist
- fehlende Einheiten viel Geld kosten können (→ [Video: NASA Measuring Failure - Mars Orbiter](#))

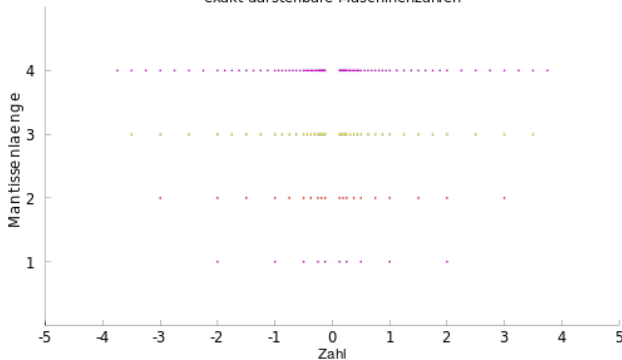
Einheiten **immer** angeben, da ...

- nie klar ist, welche Einheit gemeint ist
- es ein richtig ärgerlicher Fehler ist
- fehlende Einheiten viel Geld kosten können (→ [Video: NASA Measuring Failure](#) - Mars Orbiter)

Attribute

- Attribute sind Eigenschaften eines Objekts oder einer Klasse
- Attribute sind keine Hilfsvariablen

Nicht jede Zahl kann als Gleitkomma-Zahl dargestellt werden:
exakt darstellbare Maschinenzahlen



Quelle: de.wikipedia.org/wiki/Datei:Gleitkommazahlen.svg

Zusätzlich sollte man bei der Camera-Aufgabe von $\varepsilon = 10^6$ ausgehen.
Die Vergleiche müssen also etwa so aussehen:

```
/**
 * Check two doubles for equality.
 *
 * @param fp1 first floating point number
 * @param fp2 second floating point number
 * @return {@code true} if both floats are equal, otherwise {@code false}
 */
private boolean fpEquals(final double fp1, final double fp2) {
    return Math.abs(fp1 - fp2) < EPSILON;
}
```

Camera.java: isLeftContrastHigher

```
1  /**
2   * Determine if the contrast on the left is higher than on the current
3   * position.
4   *
5   * @param objective the objective you are manipulating
6   * @return {@code true} the contrast on the left of the current position is
7   *         higher, otherwise {@code false}
8   */
9  private boolean isLeftContrastHigher(Objective objective) {
10     double contrast = objective.getContrast();
11     objective.stepLeft();
12     double contrastNew = objective.getContrast();
13     objective.stepRight();
14
15     // check if the contrast - according to our EPSILON - is the same
16     if (fpEquals(contrast, contrastNew)) {
17         return false;
18     }
19
20     return contrastNew > contrast;
21 }
```

Camera.java: Eigentlicher Code

```
1  /**
2   * Adjust objective to get the optimum focus. The optimum focus is
3   * determined by the highest contrast.
4   */
5  public void autofocus() {
6      boolean stepLeft;
7      double contrast = objective.getContrast();
8
9      // determine direction
10     stepLeft = isLeftContrastHigher(objective);
11
12     // loop until optimum passed
13     while (objective.getContrast() > contrast
14         && !fpEquals(contrast, objective.getContrast())) {
15         contrast = objective.getContrast();
16         if (stepLeft) {
17             objective.stepLeft();
18         } else {
19             objective.stepRight();
20         }
21     }
22
23     // optional correction-move back
24     if (!fpEquals(contrast, objective.getContrast())) {
25         if (stepLeft) {
26             objective.stepRight();
27         } else {
28             objective.stepLeft();
29         }
30     }
31 }
```

Bringt es einen Vorteil, eine schwere Aufgabe nicht zu bearbeiten?

- Es gibt für jede Teilaufgabe ein Punktekontingent
- Bearbeitet ihr eine Teilaufgabe nicht, ziehe ich alle Punkte ab
- Bearbeitet ihr einen Teil einer Teilaufgabe nicht, ziehe ich mindestens so viele Punkte ab, wie derjenige mit dem höchstem Abzug abgezogen bekommen hat
- Wenn der Code nicht das tut, was gefordert wird, kann ich auch alles abziehen

⇒ Nein, es bringt keinen Vorteil

Bringt es einen Vorteil, eine schwere Aufgabe nicht zu bearbeiten?

- Es gibt für jede Teilaufgabe ein Punktekontingent
- Bearbeitet ihr eine Teilaufgabe nicht, ziehe ich alle Punkte ab
- Bearbeitet ihr einen Teil einer Teilaufgabe nicht, ziehe ich mindestens so viele Punkte ab, wie derjenige mit dem höchstem Abzug abgezogen bekommen hat
- Wenn der Code nicht das tut, was gefordert wird, kann ich auch alles abziehen

⇒ Nein, es bringt keinen Vorteil

Bringt es einen Vorteil, eine schwere Aufgabe nicht zu bearbeiten?

- Es gibt für jede Teilaufgabe ein Punktekontingent
- Bearbeitet ihr eine Teilaufgabe nicht, ziehe ich alle Punkte ab
- Bearbeitet ihr einen Teil einer Teilaufgabe nicht, ziehe ich mindestens so viele Punkte ab, wie derjenige mit dem höchstem Abzug abgezogen bekommen hat
- Wenn der Code nicht das tut, was gefordert wird, kann ich auch alles abziehen

⇒ Nein, es bringt keinen Vorteil

Bringt es einen Vorteil, eine schwere Aufgabe nicht zu bearbeiten?




- Es gibt für jede Teilaufgabe ein Punktekontingent
- Bearbeitet ihr eine Teilaufgabe nicht, ziehe ich alle Punkte ab
- Bearbeitet ihr einen Teil einer Teilaufgabe nicht, ziehe ich mindestens so viele Punkte ab, wie derjenige mit dem höchstem Abzug abgezogen bekommen hat
- Wenn der Code nicht das tut, was gefordert wird, kann ich auch alles abziehen

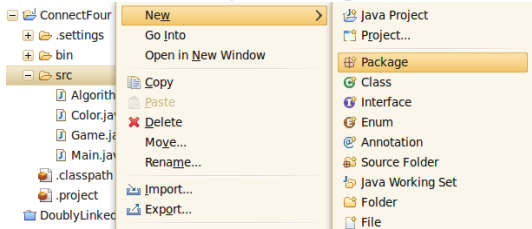
⇒ Nein, es bringt keinen Vorteil

Bringt es einen Vorteil, eine schwere Aufgabe nicht zu bearbeiten?

- Es gibt für jede Teilaufgabe ein Punktekontingent
- Bearbeitet ihr eine Teilaufgabe nicht, ziehe ich alle Punkte ab
- Bearbeitet ihr einen Teil einer Teilaufgabe nicht, ziehe ich mindestens so viele Punkte ab, wie derjenige mit dem höchstem Abzug abgezogen bekommen hat
- Wenn der Code nicht das tut, was gefordert wird, kann ich auch alles abziehen

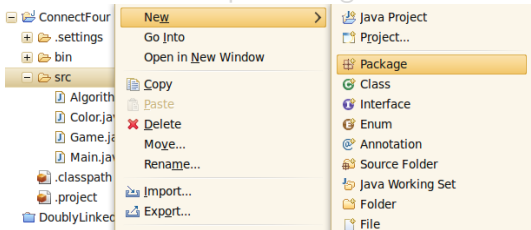
⇒ Nein, es bringt keinen Vorteil

- Dienen der Strukturierung des Quelltextes
- Sollen Namenskonflikte vermeiden (z.B. Klasse `Person` in einem Paket `uni` ist wohl anders als Klasse `Person` im Paket `politik`)
- Können über `.jar`-Dateien eingebunden werden
- Könnt ihr über  +  +  und `Package` oder mit einem Rechtsklick in Eclipse erzeugen:



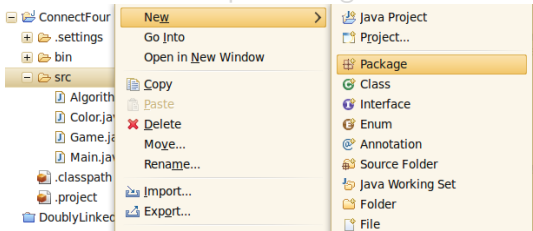
- Eine Klasse ist Teil eines Paketes, wenn sie ...
 - in dem Ordner mit dem Paketnamen liegt
 - sie `package <paketname>;` ganz am Anfang stehen hat

- Dienen der Strukturierung des Quelltextes
- Sollen Namenskonflikte vermeiden (z.B. Klasse `Person` in einem Paket `uni` ist wohl anders als Klasse `Person` im Paket `politik`)
- Können über `.jar`-Dateien eingebunden werden
- Könnt ihr über `⬆` + `Alt` + `N` und `Package` oder mit einem Rechtsklick in Eclipse erzeugen:



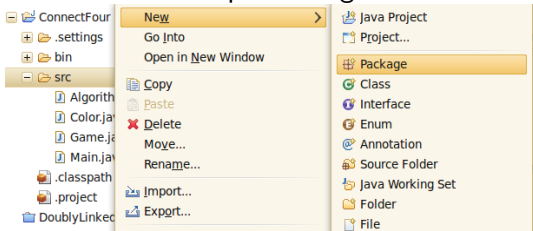
- Eine Klasse ist Teil eines Paketes, wenn sie ...
 - in dem Ordner mit dem Paketnamen liegt
 - sie `package <paketname>;` ganz am Anfang stehen hat

- Dienen der Strukturierung des Quelltextes
- Sollen Namenskonflikte vermeiden (z.B. Klasse `Person` in einem Paket `uni` ist wohl anders als Klasse `Person` im Paket `politik`)
- Können über `.jar`-Dateien eingebunden werden
- Könnt ihr über `⬆` + `Alt` + `N` und `Package` oder mit einem Rechtsklick in Eclipse erzeugen:



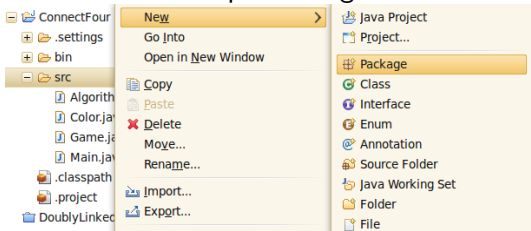
- Eine Klasse ist Teil eines Paketes, wenn sie ...
 - in dem Ordner mit dem Paketnamen liegt
 - sie `package <paketname>;` ganz am Anfang stehen hat

- Dienen der Strukturierung des Quelltextes
- Sollen Namenskonflikte vermeiden (z.B. Klasse `Person` in einem Paket `uni` ist wohl anders als Klasse `Person` im Paket `politik`)
- Können über `.jar`-Dateien eingebunden werden
- Könnt ihr über $\uparrow + \text{Alt} + \text{N}$ und `Package` oder mit einem Rechtsklick in Eclipse erzeugen:



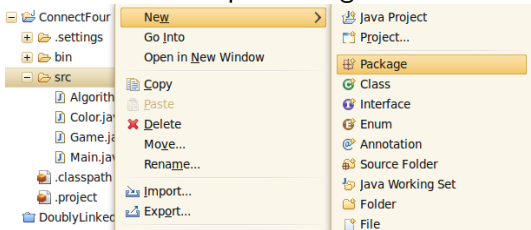
- Eine Klasse ist Teil eines Paketes, wenn sie ...
 - in dem Ordner mit dem Paketnamen liegt
 - sie `package <paketname>;` ganz am Anfang stehen hat

- Dienen der Strukturierung des Quelltextes
- Sollen Namenskonflikte vermeiden (z.B. Klasse `Person` in einem Paket `uni` ist wohl anders als Klasse `Person` im Paket `politik`)
- Können über `.jar`-Dateien eingebunden werden
- Könnt ihr über `⬆` + `Alt` + `N` und `Package` oder mit einem Rechtsklick in Eclipse erzeugen:



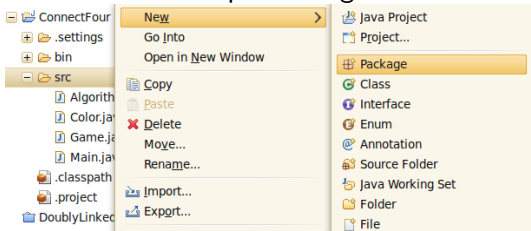
- Eine Klasse ist Teil eines Paketes, wenn sie ...
 - in dem Ordner mit dem Paketnamen liegt
 - sie `package <paketname>;` ganz am Anfang stehen hat

- Dienen der Strukturierung des Quelltextes
- Sollen Namenskonflikte vermeiden (z.B. Klasse `Person` in einem Paket `uni` ist wohl anders als Klasse `Person` im Paket `politik`)
- Können über `.jar`-Dateien eingebunden werden
- Könnt ihr über `⬆` + `Alt` + `N` und `Package` oder mit einem Rechtsklick in Eclipse erzeugen:



- Eine Klasse ist Teil eines Paketes, wenn sie ...
 - in dem Ordner mit dem Paketnamen liegt
 - sie `package <paketname>;` ganz am Anfang stehen hat

- Dienen der Strukturierung des Quelltextes
- Sollen Namenskonflikte vermeiden (z.B. Klasse `Person` in einem Paket `uni` ist wohl anders als Klasse `Person` im Paket `politik`)
- Können über `.jar`-Dateien eingebunden werden
- Könnt ihr über $\uparrow + \text{Alt} + \text{N}$ und `Package` oder mit einem Rechtsklick in Eclipse erzeugen:



- Eine Klasse ist Teil eines Paketes, wenn sie ...
 - in dem Ordner mit dem Paketnamen liegt
 - sie `package <paketname>;` ganz am Anfang stehen hat

Namenskonventionen

- Pakete werden klein geschrieben
- Pakete können „Unterpakete“ haben. Dies wird durch einen Punkt angedeutet:
 - Offizielle Pakete:
 - `java.io`
 - `java.lang` - hat z.B. `Byte`, `Integer`, ...
 - `java.lang.annotation`

Pakete: protected

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

Weitere Informationen: [Controlling Access to Members of a Class](#)

Tipp

private macht fast immer Sinn. Wenn ihr nicht wisst, ob ihr private oder protected nehmen sollt, nehmt protected. Kein modifier macht selten Sinn. Das sieht so aus, als ob ihr es dem Zufall überlasst.

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

Weitere Informationen: [Controlling Access to Members of a Class](#)

Tipp

`private` macht fast immer Sinn. Wenn ihr nicht wisst, ob ihr `private` oder `protected` nehmen sollt, nehmt `protected`. Kein modifier macht selten Sinn. Das sieht so aus, als ob ihr es dem Zufall überlasst.

Foreach ...

- wird in Java-Kreisen manchmal auch „Enhanced for loop“ genannt
- geht alle Elemente einer [Collection](#) durch zu gehen (genauer: [Iterable](#))
- sollte verwendet werden, wenn man es verwenden kann
- ist auch in der [Dokumentation](#) (Nur in 1.5?) und im [Tutorial](#)
- ist in der [JLS SE7](#) spezifiziert
- ist Teil des [Programmer Level I Exams](#) für das „Programmer Language Certification“

Foreach ...

- wird in Java-Kreisen manchmal auch „Enhanced for loop“ genannt
- geht alle Elemente einer **Collection** durch zu gehen (genauer: **Iterable**)
- sollte verwendet werden, wenn man es verwenden kann
- ist auch in der **Dokumentation** (Nur in 1.5?) und im **Tutorial**
- ist in der **JLS SE7** spezifiziert
- ist Teil des **Programmer Level I Exams** für das „Programmer Language Certification“

Foreach ...

- wird in Java-Kreisen manchmal auch „Enhanced for loop“ genannt
- geht alle Elemente einer **Collection** durch zu gehen (genauer: **Iterable**)
- sollte verwendet werden, wenn man es verwenden kann
- ist auch in der **Dokumentation** (Nur in 1.5?) und im **Tutorial**
- ist in der **JLS SE7** spezifiziert
- ist Teil des **Programmer Level I Exams** für das „Programmer Language Certification“

Foreach ...

- wird in Java-Kreisen manchmal auch „Enhanced for loop“ genannt
- geht alle Elemente einer **Collection** durch zu gehen (genauer: **Iterable**)
- sollte verwendet werden, wenn man es verwenden kann
- ist auch in der **Dokumentation** (Nur in 1.5?) und im **Tutorial**
- ist in der **JLS SE7** spezifiziert
- ist Teil des **Programmer Level I Exams** für das „Programmer Language Certification“

Foreach ...

- wird in Java-Kreisen manchmal auch „Enhanced for loop“ genannt
- geht alle Elemente einer **Collection** durch zu gehen (genauer: **Iterable**)
- sollte verwendet werden, wenn man es verwenden kann
- ist auch in der **Dokumentation** (Nur in 1.5?) und im **Tutorial**
- ist in der **JLS SE7** spezifiziert
- ist Teil des **Programmer Level I Exams** für das „Programmer Language Certification“

Foreach ...

- wird in Java-Kreisen manchmal auch „Enhanced for loop“ genannt
- geht alle Elemente einer [Collection](#) durch zu gehen (genauer: [Iterable](#))
- sollte verwendet werden, wenn man es verwenden kann
- ist auch in der [Dokumentation](#) (Nur in 1.5?) und im [Tutorial](#)
- ist in der [JLS SE7](#) spezifiziert
- ist Teil des [Programmer Level I Exams](#) für das „Programmer Language Certification“

foreach in Java: Beispiel

```
int[] myArray = { 1, 5, 6, 23, 4, 2, -1, 4 };
```

```
// for-Schleife
```

```
for (int i = 0; i < myArray.length; i++) {  
    System.out.println(myArray[i]);  
}
```

```
// foreach-Schleife
```

```
for (int element : myArray) {  
    System.out.println(element);  
}
```

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
- **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
- JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
- Pakete sollten nun verwendet werden

⇒ Achtung beim Upload in den Praktomaten! vgl. [tutorium-05.pdf](#), Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier (`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
- **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
- JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
- Pakete sollten nun verwendet werden

⇒ Achtung beim Upload in den Praktomaten! vgl. [tutorium-05.pdf](#), Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier (`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
- **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
- JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
- Pakete sollten nun verwendet werden

⇒ Achtung beim Upload in den Praktomaten! vgl. [tutorium-05.pdf](#), Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier (`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
- **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
- JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
- Pakete sollten nun verwendet werden

⇒ Achtung beim Upload in den Praktomaten! vgl. [tutorium-05.pdf](#), Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier (`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
 - **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
 - JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
 - Pakete sollten nun verwendet werden
- ⇒ Achtung beim Upload in den Praktomaten! vgl. [tutorium-05.pdf](#), Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier
(`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
 - **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
 - JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
 - Pakete sollten nun verwendet werden
- ⇒ Achtung beim Upload in den Praktomaten! vgl. [tutorium-05.pdf](#), Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier
(`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
 - **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
 - JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
 - Pakete sollten nun verwendet werden
- ⇒ Achtung beim Upload in den Praktomaten! vgl. `tutorium-05.pdf`, Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier
(`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
 - **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
 - JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
 - Pakete sollten nun verwendet werden
- ⇒ Achtung beim Upload in den Praktomaten! vgl. `tutorium-05.pdf`, Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier
(`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
 - **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
 - JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
 - Pakete sollten nun verwendet werden
- ⇒ Achtung beim Upload in den Praktomaten! vgl. `tutorium-05.pdf`, Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier
(`private`, `public`, `protected`) verwendet, soll natürlich direkt zum
jeweiligen modifier und NICHT in eine extra Textdatei!

- Nur Klassen und Methoden aus `java.lang` sind erlaubt
 - **Keine** Textausgabe und keine Schreibzugriffe auf Dateien
 - JavaDoc und Datenkapselung (→ `private`) sind nun Pflicht
 - Pakete sollten nun verwendet werden
- ⇒ Achtung beim Upload in den Praktomaten! vgl. [tutorium-05.pdf](#), Folien 23-24

A.1.1 Nur die `statements.csv` als Abgabe!

A.2 `StrangeClass.java` soll wirklich **nirgends** „f“ und kein „d“ haben

A.3 Levenshtein-Distanz → nächste Folie

A.4 Geometrie → übernächste Folie

B.1 Der Kommentar, warum ihr die Modifier (`private`, `public`, `protected`) verwendet, soll natürlich direkt zum jeweiligen modifier und NICHT in eine extra Textdatei!

- Wer hat sichs angeschaut?
- Verstanden? Falls nicht sind folgende Seiten einen Versuch wert:
 - [Guter Artikel](#), aber nur bis „Some Code, Finally“ relevant
 - [de-Wikipedia](#)
 - [en-Wiki mit Pseudocode](#)
 - [eine Visualisierung](#)

Achtung!

Der Algorithmus muss modifiziert werden. Durch die Modifikationen entspricht der Algorithmus nicht mehr den Varianten, die sich im Internet finden lassen. Ohne diese Modifikationen kann ich euch keine Punkte geben!

- Wer hat sichs angeschaut?
- Verstanden? Falls nicht sind folgende Seiten einen Versuch wert:
 - [Guter Artikel](#), aber nur bis „Some Code, Finally“ relevant
 - [de-Wikipedia](#)
 - [en-Wiki](#) mit Pseudocode
 - [eine Visualisierung](#)

Achtung!

Der Algorithmus muss modifiziert werden. Durch die Modifikationen entspricht der Algorithmus nicht mehr den Varianten, die sich im Internet finden lassen. Ohne diese Modifikationen kann ich euch keine Punkte geben!

A.3 Levenshtein-Distanz

- Wer hat sichs angeschaut?
- Verstanden? Falls nicht sind folgende Seiten einen Versuch wert:
 - [Guter Artikel](#), aber nur bis „Some Code, Finally“ relevant
 - [de-Wikipedia](#)
 - [en-Wiki](#) mit Pseudocode
 - [eine Visualisierung](#)

Achtung!

Der Algorithmus muss modifiziert werden. Durch die Modifikationen entspricht der Algorithmus nicht mehr den Varianten, die sich im Internet finden lassen. Ohne diese Modifikationen kann ich euch keine Punkte geben!

A.3 Levenshtein-Distanz

- Wer hat sichs angeschaut?
- Verstanden? Falls nicht sind folgende Seiten einen Versuch wert:
 - [Guter Artikel](#), aber nur bis „Some Code, Finally“ relevant
 - [de-Wikipedia](#)
 - [en-Wiki](#) mit Pseudocode
 - [eine Visualisierung](#)

Achtung!

Der Algorithmus muss modifiziert werden. Durch die Modifikationen entspricht der Algorithmus nicht mehr den Varianten, die sich im Internet finden lassen. Ohne diese Modifikationen kann ich euch keine Punkte geben!

A.3 Levenshtein-Distanz

- Wer hat sichs angeschaut?
- Verstanden? Falls nicht sind folgende Seiten einen Versuch wert:
 - [Guter Artikel](#), aber nur bis „Some Code, Finally“ relevant
 - [de-Wikipedia](#)
 - [en-Wiki](#) mit Pseudocode
 - [eine Visualisierung](#)

Achtung!

Der Algorithmus muss modifiziert werden. Durch die Modifikationen entspricht der Algorithmus nicht mehr den Varianten, die sich im Internet finden lassen. Ohne diese Modifikationen kann ich euch keine Punkte geben!

A.3 Levenshtein-Distanz

- Wer hat sichs angeschaut?
- Verstanden? Falls nicht sind folgende Seiten einen Versuch wert:
 - [Guter Artikel](#), aber nur bis „Some Code, Finally“ relevant
 - [de-Wikipedia](#)
 - [en-Wiki](#) mit Pseudocode
 - [eine Visualisierung](#)

Achtung!

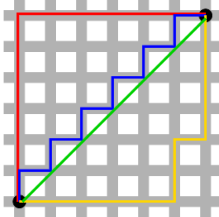
Der Algorithmus muss modifiziert werden. Durch die Modifikationen entspricht der Algorithmus nicht mehr den Varianten, die sich im Internet finden lassen. Ohne diese Modifikationen kann ich euch keine Punkte geben!

- Wer hat sichs angeschaut?
- Verstanden? Falls nicht sind folgende Seiten einen Versuch wert:
 - [Guter Artikel](#), aber nur bis „Some Code, Finally“ relevant
 - [de-Wikipedia](#)
 - [en-Wiki](#) mit Pseudocode
 - [eine Visualisierung](#)

Achtung!

Der Algorithmus muss modifiziert werden. Durch die Modifikationen entspricht der Algorithmus nicht mehr den Varianten, die sich im Internet finden lassen. Ohne diese Modifikationen kann ich euch keine Punkte geben!

■ Manhattan-Distanz:



Quelle: commons.wikimedia.org/wiki/File:Manhattan_distance.svg

Siehe Blätter 5, 6 und 7 in `prog-official` `tut_aufgaben_ws1112`

ÜB 4: Achtung

- Sehr große Aufgabe (viele Methoden / Klassen)
- Die meisten Methoden benötigen nur ein bis vier Zeilen

Themenvorschläge fürs nächste mal

- Assertions
- Verkettete Listen
- Allgemeine Übungsaufgaben

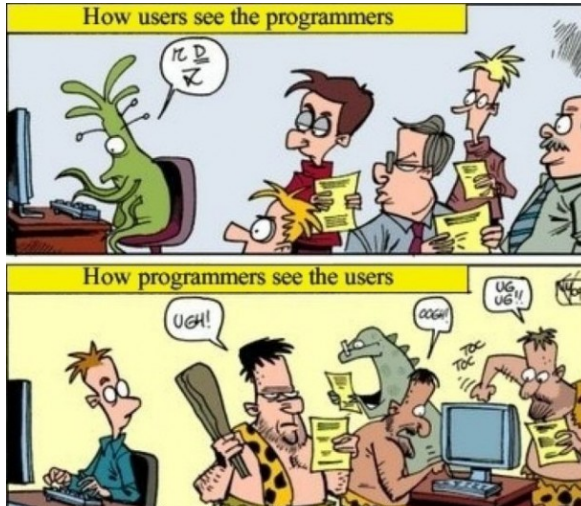
- Falls ihr Nachhilfe braucht, meldet euch bei mir oder dem Übungsleiter (Florian Merz)
- Euch wird dann ein Nachhilfelehrer vermittelt
- Die Kosten müsst ihr mit dem Nachhilfelehrer aushandeln

- Falls ihr Nachhilfe braucht, meldet euch bei mir oder dem Übungsleiter (Florian Merz)
- Euch wird dann ein Nachhilfelehrer vermittelt
- Die Kosten müsst ihr mit dem Nachhilfelehrer aushandeln

- Falls ihr Nachhilfe braucht, meldet euch bei mir oder dem Übungsleiter (Florian Merz)
- Euch wird dann ein Nachhilfelehrer vermittelt
- Die Kosten müsst ihr mit dem Nachhilfelehrer aushandeln

- 8. 26.11.2012
- 7. 03.12.2012
- 6. 10.12.2012
- 5. 17.12.2012: Video „Library“ zeigen
 - 24.12.2012: Heiligabend - **Kein Tutorium**
 - 31.12.2012: Silvester - Kein Tutorium
- 4. 07.01.2013
- 3. 14.01.2013
- 2. 21.01.2013
- 1. 28.01.2013: Abschlussprüfunsvorbereitung
- 0. 04.02.2013: Abschlussprüfunsvorbereitung
 - 10.02.2013: Ende der Vorlesungszeit des WS 2012/2013 (**Quelle**)

Vielen Dank für eure Aufmerksamkeit!



http://24.media.tumblr.com/tumblr_lv8i3tDjlk1qig5tho1_1280.jpg