



FACHSCHAFT INFORMATIK
HS Karlsruhe

Programmiervorkurs

Tag 4

Joachim Leiser

Ablauf

- 09:30 Uhr Besprechung der Übungen von Tag 3
- 10:00 Uhr Vorlesung
- 11:30 Uhr Mittagspause
 - 60 Minuten
- Gegen 12:30 Uhr Übungen im LI 136 u. LI 137
- Danach ein Bierchen?!

Inhaltsübersicht Vorkurs

- Tag 1: Zustände, Variablen, Datentypen, Konvertierungen, Arithmetik, Eclipse Live-Demo
- Tag 2: Kommentare, Boolesche Ausdrücke, If-Abfragen, Switch-Case
- Tag 3: Arrays, (Do-)While-Schleife, For-Schleifen, Weiterführung Debugging
- Tag 4: **(statische) Methoden, Klassenvariablen, JavaDoc**

Methoden



- Definition Methode: Eine Methode ist ein mehr oder weniger planmäßiges Verfahren zur Erreichung eines Zieles.
- Idee: Wir beschreiben einen planmäßigen Ablauf, eine sogenannte Methode, die wir wiederverwenden können.
- D.h. wir könnten eine Methode schreiben, die für uns die aktuelle Uhrzeit ausgibt. Und diese Methode können wir einfach mit dem Aufruf des Namens ausführen.
- Wenn die Methode existiert können wir sie also beliebig oft wiederverwenden.

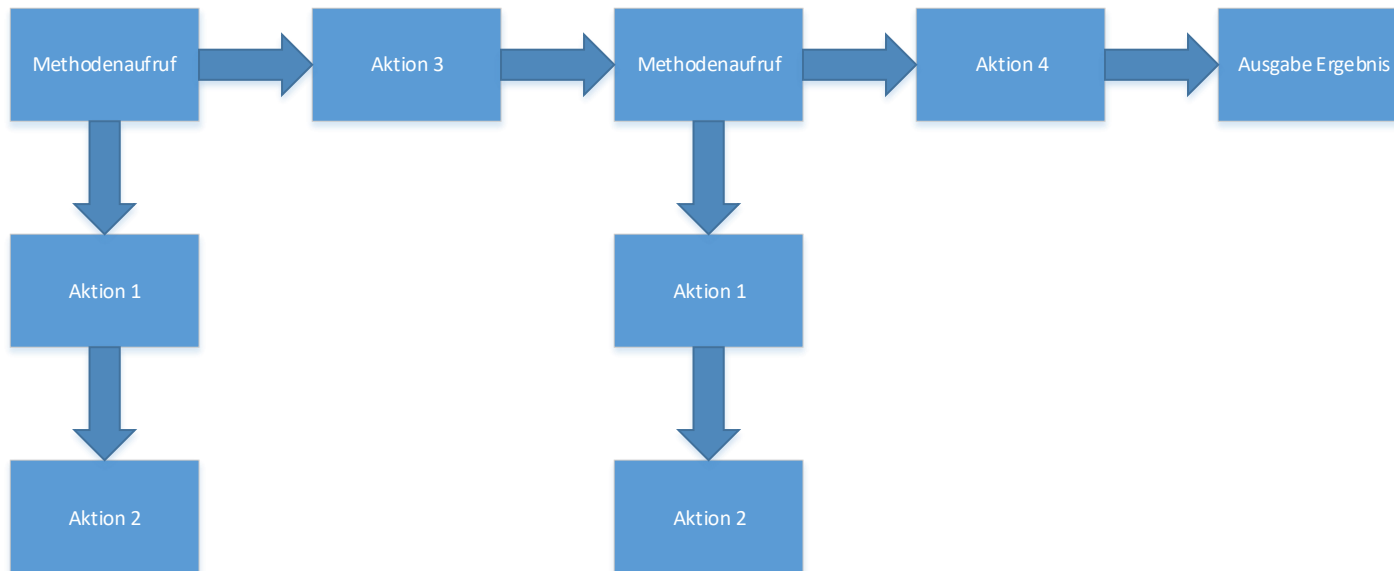
Ohne Methode

- Wir haben hier mehrere Aktionen die wir hintereinander ausführen um ein Ergebnis zu erreichen.
- Jede Aktion entspricht dabei einer Zeile Code.
- Auffällig ist, dass sich Aktionen 1 und 2 wiederholen, wir können hier aber keine Schleife verwenden.



Mit Methode

- Hier lagern wir Aktionen 1 und 2 in eine Methode aus. Wir müssen die Methode, also Aktionen 1 und 2, nur einmal schreiben.



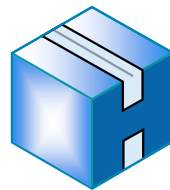
- Wir müssen jetzt jede Aktion nur noch einmal schreiben!

Beispiel println()

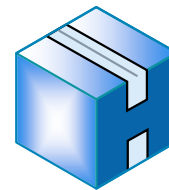
- `System.out.println("Diese Zeile enthält einen Methodenaufruf");`
- Ohne es zu wissen haben wir Methoden schon verwendet. `System.out.println()` ist nämlich ein Methodenaufruf.
- Dabei wissen wir gar nicht, wie die Methode genau die Ausgabe erzeugt. Wir wissen allerdings, dass sie uns eine Ausgabe erzeugt. Mehr brauchen wir ja auch nicht wissen.
- Man nennt das auch Entkopplung der Funktionalität.

Klassen und Methoden

- Um richtig zu verstehen wie Methoden funktionieren müssen wir uns dabei auch mit Klassen auseinandersetzen.
- Wir können uns Klassen fürs erste wie Pakete/Kartons vorstellen.



Test

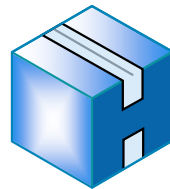


Hilfe

- Wir haben nun eine Klasse namens Test und eine namens Hilfe. In der Klasse Test schreiben wir unseren Code, in der Klasse Hilfe hat jemand bereits für uns Code geschrieben, den wir verwenden dürfen.

Klassen und Methoden

- Wir können uns eine Methode als einen kleinen Arbeiter vorstellen der eine beschriebene Aufgabe abarbeiten kann.
- Dieser Arbeiter sitzt in einem unserer Pakete. In unserem Fall sitzt er im Paket Hilfe. Er hat den Namen „sayHello“ und seine Aufgabe ist es „Hello World“ zu sagen.



Test



Hilfe

- Wir können dem Arbeiter jetzt jederzeit und beliebig oft sagen, dass er seine Aufgabe ausführen soll.
- Wenn wir uns also Arbeiter bzw. Methoden für alle möglichen Aufgaben schreiben, könnte das unser Leben viel leichter machen.

Methoden aufrufen

- Wir schreiben folgenden Code:

```
public class Test {  
    public static void main(String[] args) {  
        Hilfe.sayHello();  
    }  
}
```

- Dieser Code gibt uns „Hello World“ aus, denn wir haben eine Methode (bzw. Arbeiter) der Klasse (bzw. des Pakets) Hilfe namens sayHello() aufgerufen die eben dies für uns macht.
- Ein Methodenaufruf geht also so:

```
Klassenname.methodName();
```

Methoden schreiben

- Die zugehörige Klasse (bzw. Paket) Hilfe sieht dann so aus:

```
public class Hilfe {  
    public static void sayHello() {  
        System.out.println("Hello World");  
    }  
}
```

- Hier haben wir eine Methode (bzw. Arbeiter) namens „sayHello“ geschrieben die für uns ein „Hello World“ in der Konsole ausgibt.
- In den nachfolgenden Folien werden wir die einzelnen Keywords erörtern und zeigen wie wir eine Methode nach unseren Wünschen anpassen können.

Klassen und Methoden

- Der vollständige Code:

```
public class Test {  
    public static void main(String[] args) {  
        Hilfe.sayHello();  
    }  
}  
  
public class Hilfe {  
    public static void sayHello() {  
        System.out.println("Hello World");  
    }  
}
```

Klassen und Methoden

- Wenn die Methode und Aufruf in derselben Klasse sind, kann man den Klassennamen beim Methodenaufruf weglassen:

```
public class TestMain {  
    public static void main(String[] args) {  
        sayHello();  
    }  
    public static void sayHello() {  
        System.out.println("Hello World");  
    }  
}
```

Klassen und Methoden

- Ein paar wichtige Eigenschaften von Methoden:
- Üblicherweise haben Methoden ihren eigenen Namensraum. D.h. Variablen die wir vor dem Aufruf der Methode definiert hatten sind in der Methode in der Regel nicht verfügbar.
- Genauso sind Variablen die wir in einer Methode definieren nach der Methode nicht mehr verfügbar.
- Unsere Programme werden ja Zeile für Zeile abgearbeitet. Also eine Zeile nach der anderen. Wenn wir einen Methodenaufruf haben, dann springt der „Programmzeiger“ (der anzeigt in welcher Zeile wir gerade sind) in die Methode. Wenn die Methode fertig ist springt der Programmzeiger wieder zurück in den Methodenaufruf.

Debugging

- Wie sieht es aus wenn wir dieses Programm debuggen würden?
- Beim debuggen sehen wir was das Programm eigentlich Zeile für Zeile macht. Durch das debuggen kann man ein Gefühl dafür bekommen wie sich das Programm verhält und welche Auswirkungen einzelne Zeilen haben können. Wir sehen vor allem den besagten „Programmzeiger“.
- Da die Vorführung auf Folien etwas schwierig ist beschreibe ich lediglich kurz was beim debuggen passieren würde. Beim Methodenaufruf „`Hilfe.sayHello();`“ würden wir runter in die Methode springen. Wenn die Methode fertig ausgeführt wurde springen wir wieder zurück zum Methodenaufruf.

main() Methode

- Wenn wir uns den Code genauer anschauen, dann erkennen wir das die main() die wir ständig verwenden auch eine Methode ist.
- Die main() Methode wird, nachdem der Computer den Code in Maschinensprache übersetzt hat, aufgerufen. Sie ist somit der Einstiegspunkt und hat damit unter den Methoden einen ganz besonderen Stellenwert.
- D.h. wir sollten in unserem Code die main() Methode stets an einen offensichtlichen Punkt setzen.

Methoden: Parameter

- Denken wir zurück an unseren Arbeiter der eine Methode symbolisiert.
- Unser Arbeiter führt für uns eine bestimmte Aufgabe aus und keine andere. Dabei können wir bisher die Aufgabe nicht beeinflussen. Der Arbeiter macht stupide seine beschriebene Arbeit.
- In der Metapher unseres Arbeiters wäre es doch nur logisch, wenn wir beim Aufruf unserem Arbeiter sagen könnten welche Werkzeuge oder Materialien er für seine Arbeit nehmen soll.
- Wir könnten also z.B. einen Arbeiter haben der für uns zwei Zahlen multipliziert. Diese können wir ihm einfach beim Aufruf mitteilen.
- Diese „Zahlen“ nennen wir Parameter.

Methoden: Parameter

- Eine Methode mit Parametern kann so aussehen:

```
public static void multiplyNumbers(int a, int b) {  
    System.out.println(a * b);  
}
```

- D.h. in den runden Klammern, die wir bisher leer gelassen hatten können wir Parameter eintragen.
- Ein Parameter hat einen Datentyp und einen Namen.
- Mehrere Parameter werden mit Komma getrennt. Wir können beliebig viele Parameter angeben mit verschiedenen Datentypen.
- Ein Parameter funktioniert quasi wie eine Variable.

Methoden: Parameter

- Der zugehörige Aufruf könnte so aussehen:

```
int number = 268;
```

```
multiplyNumbers(number, 144);
```

- Wir können also sowohl Variablen als auch Zahlen beim Aufruf angeben.
- Zu beachten ist, dass die Werte der Parameter bei Datentypen kopiert werden. D.h. wenn wir die Parameter innerhalb der Methode verändern, werden die Variablen die wir übergeben hatten nicht verändert.

Methoden: Rückgabewerte

- Bisher haben wir unsere Ergebnisse direkt in der Konsole mithilfe von „`System.out.println()`“ ausgegeben.
- Die Konsole ist aber eigentlich nur eine Ein- und Ausgabe für den Anwender des Programms das wir schreiben.
- Wir haben von Programmiererseite bisher keinen Zugriff auf das Ergebnis einer Methode, also wir können das Ergebnis einer Methode nicht im Code weiterverwenden.
- Das wollen wir jetzt ändern, schauen wir uns noch einmal das Beispiel von gerade eben an:

```
public static void multiplyNumbers(int a, int b) {  
    System.out.println(a * b);  
}
```

Methoden: Rückgabewerte

```
public static void multiplyNumbers(int a, int b) {  
    System.out.println(a * b);  
}
```

- Wir haben einige der Begriffe die in unserer Methode vorkommen noch gar nicht erklärt. Das „**void**“ (engl.: „leer“) sagt aus dass diese Methode keinen Rückgabewert hat (einen leeren Rückgabewert).
- Wir können das „**void**“ jetzt durch einen beliebigen Datentyp austauschen. Z.B. könnte unsere Methode jetzt so aussehen:

```
public static int multiplyNumbers(int a, int b) {  
    return (a * b);  
}
```

Methoden: Rückgabewerte

```
public static int multiplyNumbers(int a, int b) {  
    return (a * b);  
}
```

- Statt „**void**“ steht jetzt „**int**“ im Methodenkopf. D.h. unsere Methode gibt als Ergebnis einen Wert des Datentyps „**int**“, also eine Zahl, zurück.
- Außerdem haben wir das „*System.out.println(a * b);*“ durch ein „**return (a * b);**“ ersetzt. „**return**“ bedeutet dass an dieser Stelle die Methode endet und den nachfolgenden Wert zurückgibt. In unserem Fall wird also das Ergebnis der Multiplikation von **a** und **b** zurückgegeben.

Methoden: Rückgabewerte

- Der gesamte Code mit Methodenaufruf könnte also so aussehen:

```
public static int multiplyNumbers(int a, int b) {  
    return (a * b);  
}
```

```
public static void main(String[] args) {  
    int number = 11;  
    int ergebnis = multiplyNumbers(number, 144);  
    System.out.println(multiplyNumbers(12, 145));  
}
```

- Der Rückgabewert der Methode ersetzt quasi die Methode. Der Wert kann also gemäß seinem Datentyp beliebig weiter verwenden.

public/private

- Ein weiteres Keyword das wir noch nicht erklärt haben ist das „**public**“ Keyword (engl.: „öffentlich“)
- Wir haben es bisher in jeder Methodendefinition verwendet aber noch nicht geklärt was es überhaupt bedeutet.
- Es ist eigentlich ganz einfach. Methoden sind ja Teile von Klassen und das Keyword „**public**“ sagt über die Methode aus dass sie von ausserhalb (der Klasse) aufgerufen werden darf. Also darf jede andere Klasse diese Methode aufrufen.
- Wenn wir stattdessen das Keyword „**private**“ verwenden darf die Methode nicht von ausserhalb der Klasse aufgerufen werden. Sie darf nur noch von anderen Methoden derselben Klasse aufgerufen werden.

public/private

- Ein Beispiel:

```
public class Hilfe {  
    private static void ichBinGeheim() {  
        System.out.println("Ich bin geheim!");  
    }  
}
```

- Diese Methode darf nur aus der Klasse Hilfe aufgerufen werden!
- Keine andere Klasse darf diese Methode aufrufen.

public/private

- Der Nutzen dahinter ist relativ simpel. „**public**“ und „**private**“ brauchen wir hauptsächlich für die Übersichtlichkeit.
- Wir verwenden „**private**“ wo es semantisch sinnvoll ist! D.h. beim programmieren von Methoden wird uns manchmal auffallen dass wir die ein oder andere Methode auch ohne Probleme „**private**“ machen können.
- Dies wird vor allem notwendig wenn wir mit anderen Programmierern gemeinsam arbeiten oder die Projekte eine gewisse Größe erreichen.
- Für den Anfang könnt ihr die Methoden immer „**public**“ machen.

Static

- Auch das Keyword „**static**“ (engl.: „statisch“) verwenden wir ständig, obwohl wir noch gar nicht geklärt haben was es bedeutet.
- „**static**“ bedeutet dass die Methode immer da ist, dass die „Lebensdauer“ der Methode vom Programmstart bis –ende ist.
- Das klingt jetzt als gäbe es auch noch eine andere Möglichkeit. Ja, die gibt es. Diese werdet ihr aber erst im Verlaufe des Studiums kennen lernen.
- Für den Anfang sollt ihr alle Methoden statisch machen.

JavaDoc



- JavaDoc ist eine Möglichkeit Code zu dokumentieren. Vor allem wenn Projekte größer werden kommt es gerne vor dass wir nicht mehr genau wissen was eine Methode genau macht.
- Wir haben aus diesem Grund die Kommentarfunktion kennengelernt. Sie bietet uns die Möglichkeit Erinnerungsstützen in den Code einzubauen.
- JavaDoc macht quasi dasselbe, nur dass die Formatierung und Anzeige ein wenig ansprechender ist.
- An einem Beispiel wird das klarer:

JavaDoc

```
/**  
 * Nimmt zwei integer Zahlen a und b entgegen und  
 multipliziert diese miteinander.  
 * @param a  
 * @param b  
 * @return  
 */  
public static int multiplyNumbers(int a, int b) {  
    return (a * b);  
}
```

JavaDoc

- Mittels den Zeichen „`/**`“ wird ein JavaDoc Kommentar angefangen. Mit „`*/`“ hört er wieder auf.
- Die „`@param a`“ und „`@return`“ werden automatisch erzeugt. Man kann eine Beschreibung angeben was die Methode genau macht bzw. näheres zu den Parametern oder return Werten angeben.
- Aus JavaDoc Kommentaren lässt sich auch automatisch eine Dokumentation erstellen.
- Die Standard Library von Java ist komplett mit JavaDoc Kommentaren versehen. (Der Quelltext der Library muss zum Anzeigen der Kommentare importiert werden)
- Aber meiner Meinung nach das praktischste ist, dass man den JavaDoc Kommentar lesen kann indem man mit der Maus über einem Methodennamen (z.B. beim Methodenaufruf oder Autovervollständigung (Strg + Leertaste)).

JavaDoc

- Hier ein Beispiel wie es aussieht wenn man mit dem Mauszeiger über der Methode stehen bleibt:

```
74- /**
75  * Nimmt zwei integer Zahlen a und b entgegen und mutlipliziert diese miteinander.
76  * @param a
77  * @param b
78  * @return
79  */
80- public static int multiplyNumbers(int a, int b) {
81     return (a * b)
82 }
83
84 }
85
```

int PlaygroundMain.multiplyNumbers(int a, int b)

Nimmt zwei integer Zahlen a und b entgegen und mutlipliziert diese miteinander.

Parameters:

- a
- b

Returns:

Press 'F2' for focus

Problems @ Javadoc

No consoles to display at this ti

Vorteile Methoden

- Entkopplung der Funktionalität (Wir wissen, als Anwender, was eine Methode macht, aber nicht wie sie das macht)
 - `System.out.println(„Wir wissen nicht wie diese Methode intern funktioniert“);`
- Übersichtlichkeit
 - `startGame();`
 - `gameloop();`
 - `endGame();`
- Wiederverwendung
 - Redundanzen (doppelter Code) kann vermieden werden

Konventionen

- Methoden schreiben wir klein (lowerCamelCase)
- Eine Methode macht etwas, also enthält sie ein Verb
- Aussagekräftige Namen verwenden
- JavaDoc Beschreibung, was die Methode macht, braucht und zurückgibt
- Kopierter Code ist schlechter Code, stattdessen Methoden verwenden.
- Keep it simple!

Beispiel aus der Algebra

- Wie wäre es wenn wir eine Methode hätten die für uns berechnet ob ein Punkt auf einer Linie ist.

```
// f(x) = 2x + 1;  
// P(x|y), x = -2, y = -3  
isPointOnLine(2, 1, -2, -3);
```

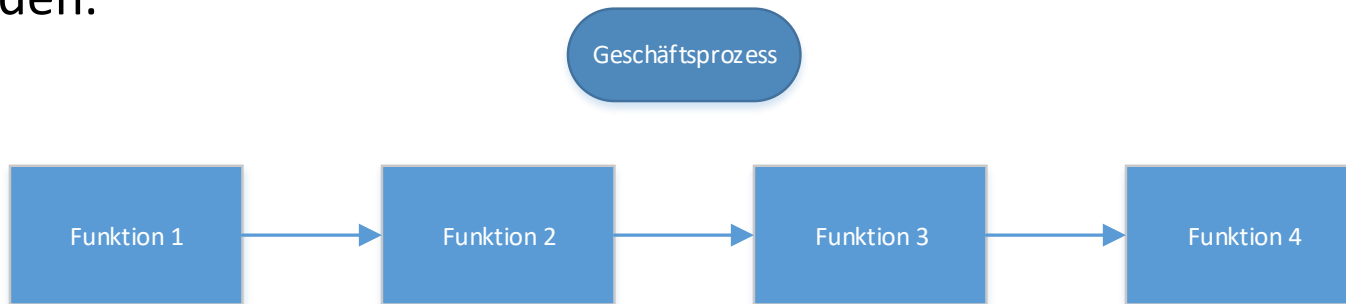
- Wir können jetzt also mit einem Punkt und einer Funktion immer berechnen ob der Punkt auf der Linie liegt.

Beispiel aus der Algebra

```
/**
 * This method checks whether the point P(x|y) is on the line
 *  $f(x) = m \cdot x + c$ .
 * @param m factor of x of line f
 * @param c offset of line f
 * @param x coordinate of the point P
 * @param y coordinate of the point P
 * @return true when P(x|y) is on f(x)
 */
public static boolean isPointOnLine(int m, int c, int x, int y) {
    //  $y = m \cdot x + c$ ;
    int result = m * x + c;
    if (result == y) {
        return true;
    }
    return false;
}
```

Beispiel aus der Praxis (abstrakt)

- Situation: Wir haben einen Geschäftsprozess, den wir mittels eines Programmes abbilden sollen. Wir können den Geschäftsprozess in verschiedene Funktionen aufteilen die nacheinander ausgeführt werden.



- → Wir können die einzelnen Funktionen jeweils als Methode schreiben. Für jede Funktion steht uns eine Aufgabenbeschreibung zur Verfügung. Wir müssten also nur die Aufgabenbeschreibung ausprogrammieren.

Klassenvariablen

- Wir kennen bereits Variablen. Bisher haben wir sie aber nur innerhalb von Methoden verwendet, also so:

```
public class Test {  
    public static void main(String[] args) {  
        int x = 3;  
    }  
}
```

- Aber, sobald wir die main() Methode verlassen können wir nicht mehr auf die Variable x zugreifen. Die Lebenszeit der Variable x gilt nur bis zum Ende der main() Methode. Wenn wir andere Methoden aufrufen haben wir x dort ebenfalls nicht.

Klassenvariablen

- Denken wir zurück an unsere Metapher von den Paketen (Klassen) und unseren Arbeitern (Methoden). Wie würden wir Variablen in diese Metapher aufnehmen?
- Variablen wären wie kleine Container in denen wir Dinge reinlegen/speichern können. Bei unseren bisherigen Variablen werden diese innerhalb einer Methode erzeugt und spätestens am Ende der Methode wieder zerstört. D.h. normale Variablen sind nur innerhalb der Methode verwendbar in der sie erzeugt wurden.
- Klassenvariablen sind anders. Sie werden zum Programmstart erzeugt und erst zum Programmende zerstört. D.h. sie sind eigentlich immer da. Das sind also Container die über mehrere Arbeiter hinweg verwendet werden können. Sie verhalten sich also wie globale Variablen, die immer zur Verfügung stehen.

Klassenvariablen

- Hier die Definition einer Klassenvariablen x:

```
public class Hilfe {  
    public static int x = 3;  
}
```

- Auf diese Variable kann wie folgt zugegriffen werden:

```
public class TestMain {  
    public static void main(String[] args) {  
        Hilfe.x = 5;  
    }  
}
```

Klassenvariablen Beispiele

- Von wo und wie man die Variablen verwenden kann funktioniert genauso wie bei den Methoden, hier ein paar Beispiele:

```
public class Hilfe {  
  
    private static int x = 3;  
  
    public static void main(String[] args) {  
        System.out.println(x + 10);  
    }  
}
```


Klassenvariablen Beispiele

```
public class Test {  
  
    public static String welcomeSentence = "Herzlich  
Willkommen!";  
  
    public static void sayWelcome() {  
        System.out.println(welcomeSentence);  
    }  
  
}
```

```
public class MyMath {
    /**
     * Die Mathematische Konstante Pi.
     */
    public static double Pi = 3.1415;
    /**
     * Berechnet den Umfang eines Kreises.
     * @param radius
     * @return Umfang
     */
    public static double umfangBerechnen(double radius) {
        return 2 * Pi * radius;
    }
    /**
     * Berechnet den Radius eines Kreises.
     * @param Umfang
     * @return radius
     */
    public static double radiusBerechnen(double Umfang) {
        return Umfang / (2 * Pi);
    }
}
```

```
public class MyMath {
```

```
...
```

```
/**
```

```
 * Berechnet den Durchmesser eines Kreises.
```

```
 * (unter Verwendung der Methode  
radiusBerechnen(Umfang))
```

```
 * @param Umfang
```

```
 * @return Durchmesser
```

```
 */
```

```
public static double durchmesserBerechnen(double  
Umfang) {
```

```
    return radiusBerechnen(Umfang) * 2;
```

```
}
```

```
}
```

Konventionen Klassenvariablen

- lowerCamelCase
- Aussagekräftige Namen
- Nur verwenden wenn wirklich notwendig
- Keep it simple!

JavaDoc Klassenvariablen

- Das JavaDoc kann natürlich auch bei Klassenvariablen verwendet werden:

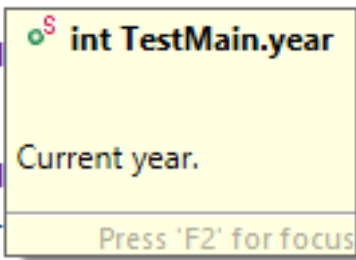
```
/**
```

```
 * Current year.
```

```
 */
```

```
public static int year = 2017;
```

```
4-  /**
5   * Current year.
6   */
7   public static int year = 2017;
8
9-  public static void main(String[] args) {
10      Hilfe.x = 5;
11  }
12-  public static void printHelp() {
13      System.out.println("Hilfe");
14  }
```



Was haben wir heute gelernt?



- Wir haben statische Methoden und ihre verschiedenen Verwendungsmöglichkeiten kennen gelernt. Mit Methoden haben wir eine schöne Möglichkeit unseren Code zu strukturieren.
- Außerdem haben wir statische Klassenvariablen kennen gelernt. Diese ergänzen unsere Methoden auf eine schöne Weise, da sie viele Problemstellungen in Kombination mit Methoden sehr vereinfachen kann.
- Zudem können wir jetzt mit JavaDoc Kommentaren unseren Code auf einfache und schöne Art beschreiben.
- Mithilfe der Werkzeuge die wir hier kennen gelernt haben lassen sich bereits sehr komplexe Programme schreiben.

Vielen Dank für eure Aufmerksamkeit
und viel Erfolg mit den Übungen