API und Kommentare

Javakurs 2014, 4. Vorlesung

Georg Hieronimus

basierend auf der Vorlage von
Theresa Enghardt, Mario Bodemann und Sebastian Dyroff

`wiki.freitagsrunde.org`

6. März 2014

Was bisher geschah...

Bisher:

„So wird das gemacht.“

- Programme schreiben, kompilieren und ausführen
- Variablen deklarieren, definieren und benutzen
- Fallunterscheidungen
- Arrays verwenden
- Schleifen
- Methoden

Was bisher geschah...

Bisher:

„So wird das gemacht.“

- Programme schreiben, kompilieren und ausführen
- Variablen deklarieren, definieren und benutzen
- Fallunterscheidungen
- Arrays verwenden
- Schleifen
- Methoden

Jetzt:

„So solltet ihr das machen.“

- Kommentare
- Lesbarkeit von Code
- Die Java API
- Komplexe Aufgabenstellungen bearbeiten
- Testen
- Externe Bibliotheken nutzen

Inhaltsverzeichnis

Inhaltsverzeichnis

Kommentare

Inlinekommentar

```
// Kommentar bis zum Ende der Zeile
```

- Oft mitten im Code zu finden
- meistens ein Hinweis, was an dieser Stelle passiert

Blockkommentar

```
/* Geht über
   mehrere Zeilen */
```

- Oft oberhalb einer Klasse oder Methode zu finden
- Meistens eine mehrzeilige Beschreibung dessen, was sie tut
- Kann außerdem Angaben zu Autor/in, Datum, Aufgabenstellung... enthalten

Anwendung von Kommentaren

Beispielklasse MatrixMax mit Kommentaren (1. Hälfte)

```
1  /* Klasse MatrixOperationen
2   * @author Georg Hieronimus <georg.hieronimus...>
3   *
4   * Stellt verschiedene statische Funktionen für Matrizen bereit.
5   */
6  public class MatrixOperationen {
7
8    /*
9    * getSpaltenMaximum
10   * Berechnet das Maximum jeder Spalte einer gegebenen Matrix
11   * und speichert die Maxima in einem neuen Array
12   *
13   * @param matrix - 4x3 Matrix
14   * @return Array mit dem Maximum der jeweiligen Spalte oder null
         bei falscher Eingabe
15   */
16   public static int[] getSpaltenMaximum(int matrix[][]) {
```

- Name des Autors
- Allgemeine Beschreibung, was die Klasse tut

Anwendung von Kommentaren

Beispielklasse MatrixMax mit Kommentaren (2. Hälfte)

```
16  public static int[] getSpaltenMaximum(int matrix[][]) {
17    //Teste die Eingabe
18    if (matrix.length != 3) return null;
19    if (matrix[0].length != 4) return null;
20
21    //Rückgabewert initialisieren
22    int maximum[] = new int[4];
23
24    for (int j = 0; j < 4; j++) {
25      int maxtemp = 0;
26      for (int i = 0; i < 3; i++) {
27        if (maxtemp < matrix[i][j]) {
28          // Neuer Kandidat ist größer - ersetze altes Maximum
29            maxtemp = matrix[i][j];
30        }
31      }
32      maximum[j] = maxtemp;//Übernehme temporäres Maximum als
            Ergebnis
33    }
34    return maximum;
```

- An Stellen, an denen „etwas passiert “

Inhaltsverzeichnis

Lesbarkeit: Formatierung

Benutzt Leerzeilen!

Lasst Platz!

Rückt ein!

Faustregel:

Innerhalb geschweifter Klammern einmal einrücken
(z.B. in Methoden, Klassen, Bedingungen, Schleifen...)

→ Man sieht besser, wo die Schleife anfängt und wo sie aufhört

- Formatierung dient nur der Lesbarkeit für euch und andere
- Der Compiler optimiert Kommentare, Leerzeilen usw. einfach weg

→ Euer Programm wird dadurch nicht langsamer!

Lesbarkeit: Einfachheit

Nachvollziehbarkeit vor Eleganz!

Ausgabe mit Fragezeichenoperator

```
System.out.println((i==4)? "i ist 4" : "i ist nicht 4");
```

Fragezeichenoperator:

bedingung ? aktion-wenn-wahr : aktion-wenn-falsch

Ausgabe mit if-then-else

```
if (i == 4) {
  System.out.println("i ist 4");
} else {
  System.out.println("i ist nicht 4");
}
```

Beide Codebeispiele machen genau das Gleiche!

Das obere ist nur schwerer zu lesen.

Lesbarkeit: Sprechende Variablennamen

Halber Satz als Klassen- oder Variablenname?

- CamelCase macht's möglich!

Gut	
Fehler `ArrayIndexOutOfBoundsException`	Index eines Arrays außerhalb der Grenzen
Methode `isCharInWord`	Ist das Zeichen im Wort enthalten?
Variable `isInWord`	Impliziert Ja/Nein-Frage $\rightarrow$ boolean

Schlecht	
Methode `convert`	Konvertiere was zu was?
Variable `foo`	Könnte alles Mögliche sein...

Lesbarkeit: Dokumentation

Dokumentation: Handbuch zum Code
Hilft anderen, aber auch dir selbst.

[1]

Read **T**he **F**ucking **M**anual - Lies das verdammte Handbuch!

(... und schreib ein verdammtes Handbuch!)

[1] http://www.flickr.com/photos/e-coli/419976117/

Inhaltsverzeichnis

Aufgabenstellung

Aufgabe: Implementiere das Spiel „Hangman“.

Schritt 1: Aufgabe verstehen

- Wie sind die Spielregeln von Hangman?
  - Buchstaben eines Wortes raten
  - Richtige werden eingetragen, falsche füllen zum „Galgen“
  - Lösen = Das ganze Wort erraten
- Was genau soll das Programm können?
  - Zufällig ein Wort auswählen
  - Auf Eingaben der Benutzerin/des Benutzers reagieren
  - Einen Galgen ausgeben
- Wie soll die Ausgabe aussehen?
  - Nur Anzahl der Striche?
  - ASCII-Art auf der Kommandozeile?
  - Grafisch?

Im Zweifelsfall nachfragen!

Ablauf aufschreiben

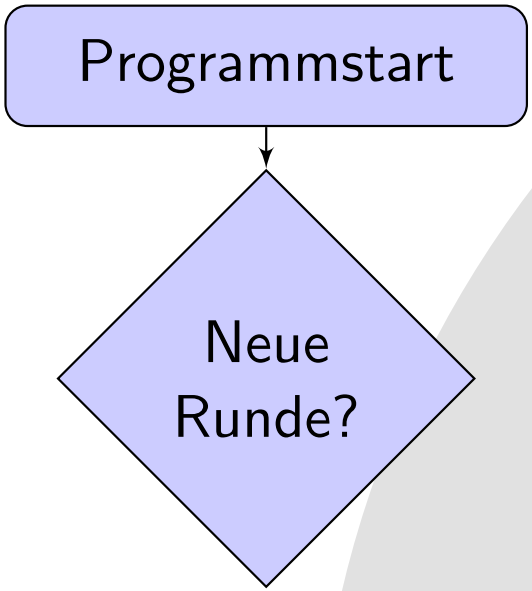
Schritt 2: Ablauf des Programms aufschreiben

- Ordnet den Ablauf
- Definiert Teilbereiche
  - Teile-und-Herrsche-Prinzip
  - Nützlich für Gruppenarbeiten
  - z.B. Ein- und Ausgabe, Wörterbearbeitung
- Am besten grafisch
  - Mit Stift und Papier
  - Am Rechner (Flowchart)

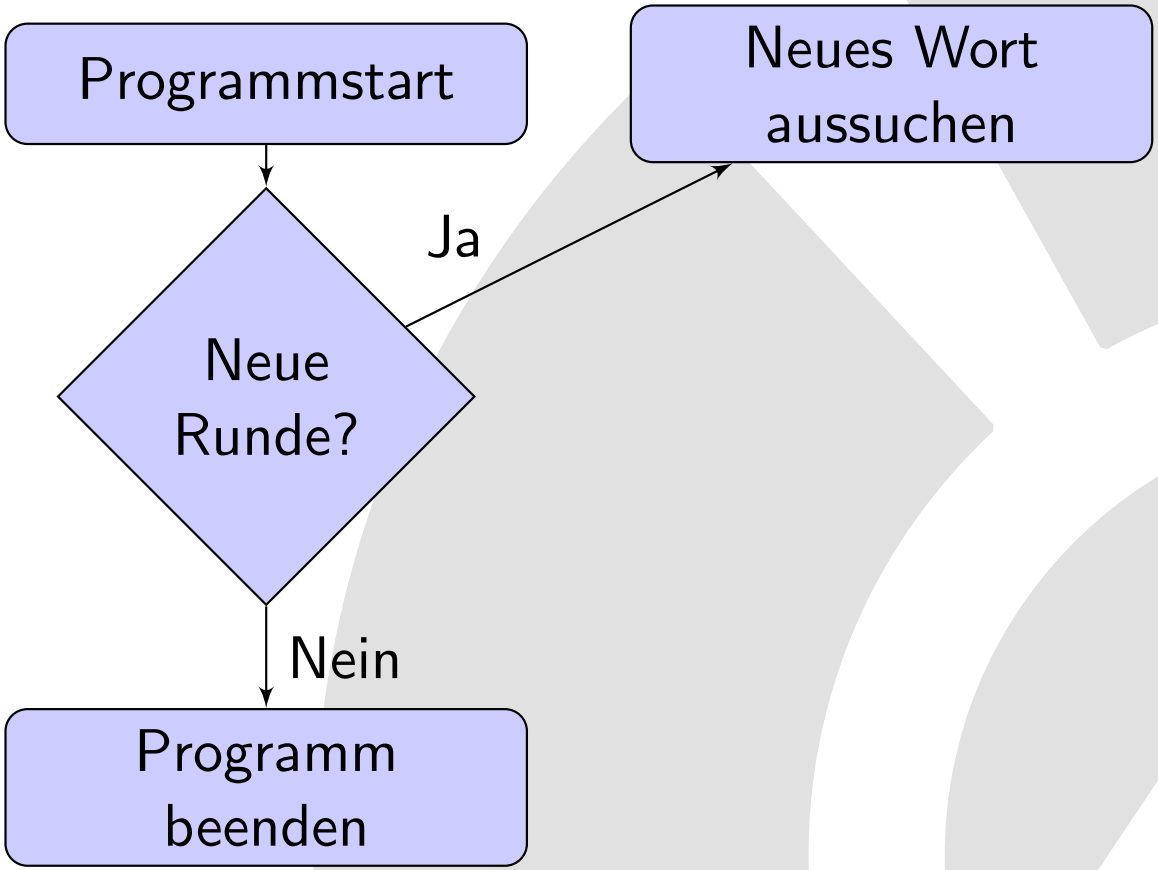
[2]

[2] http://www.flickr.com/photos/stevegarfield/616793140

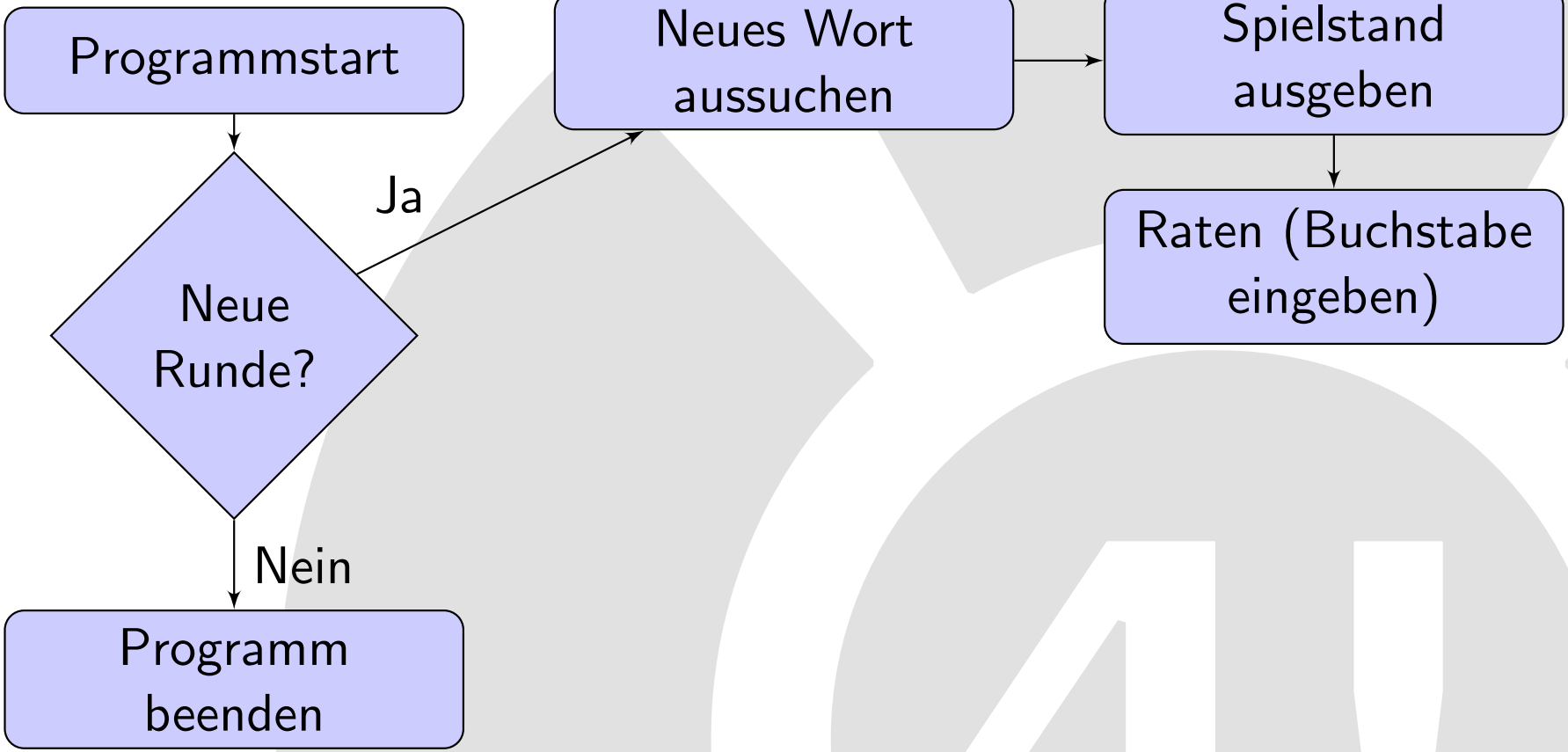
Ablauf aufschreiben



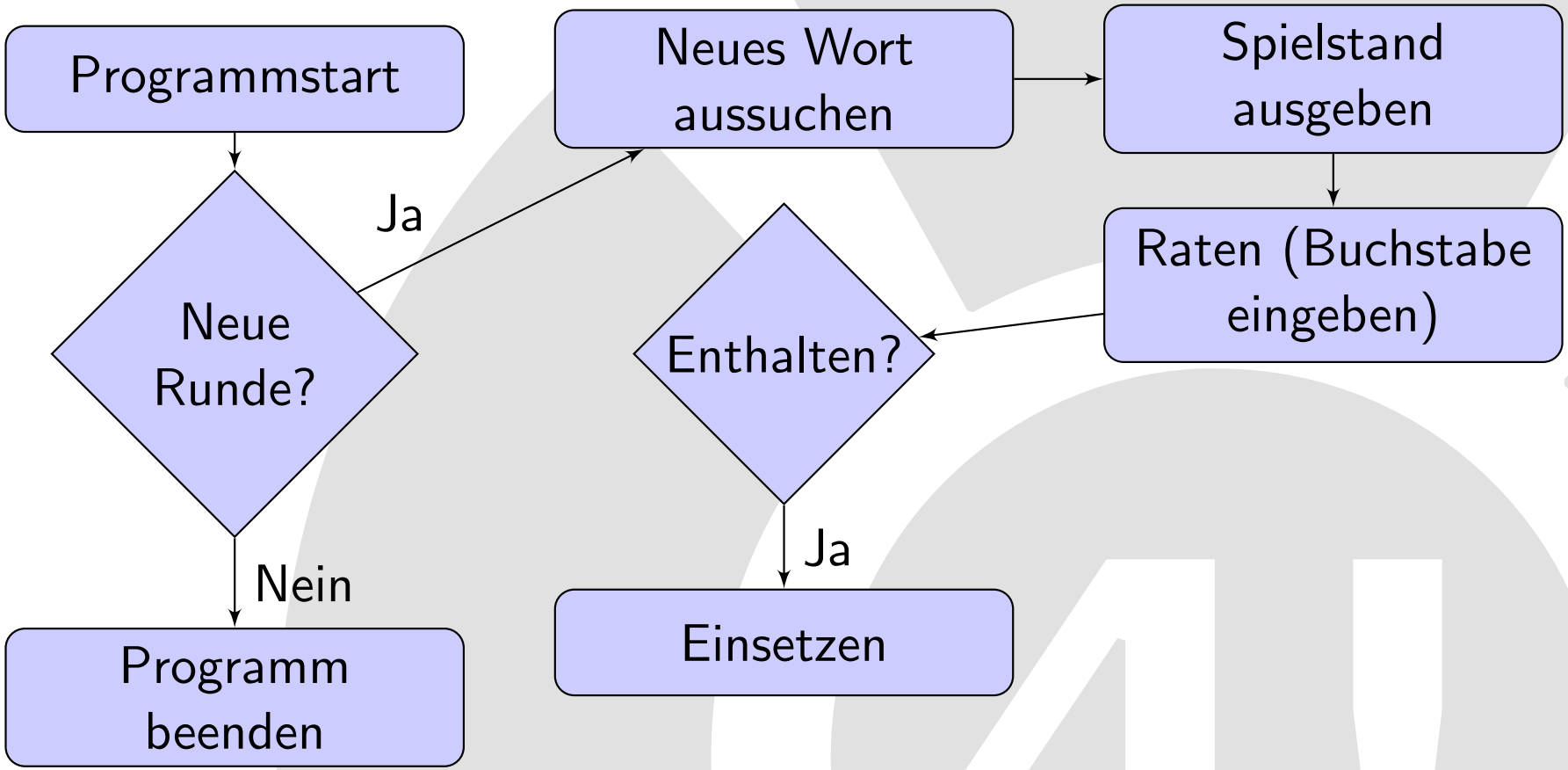
Ablauf aufschreiben



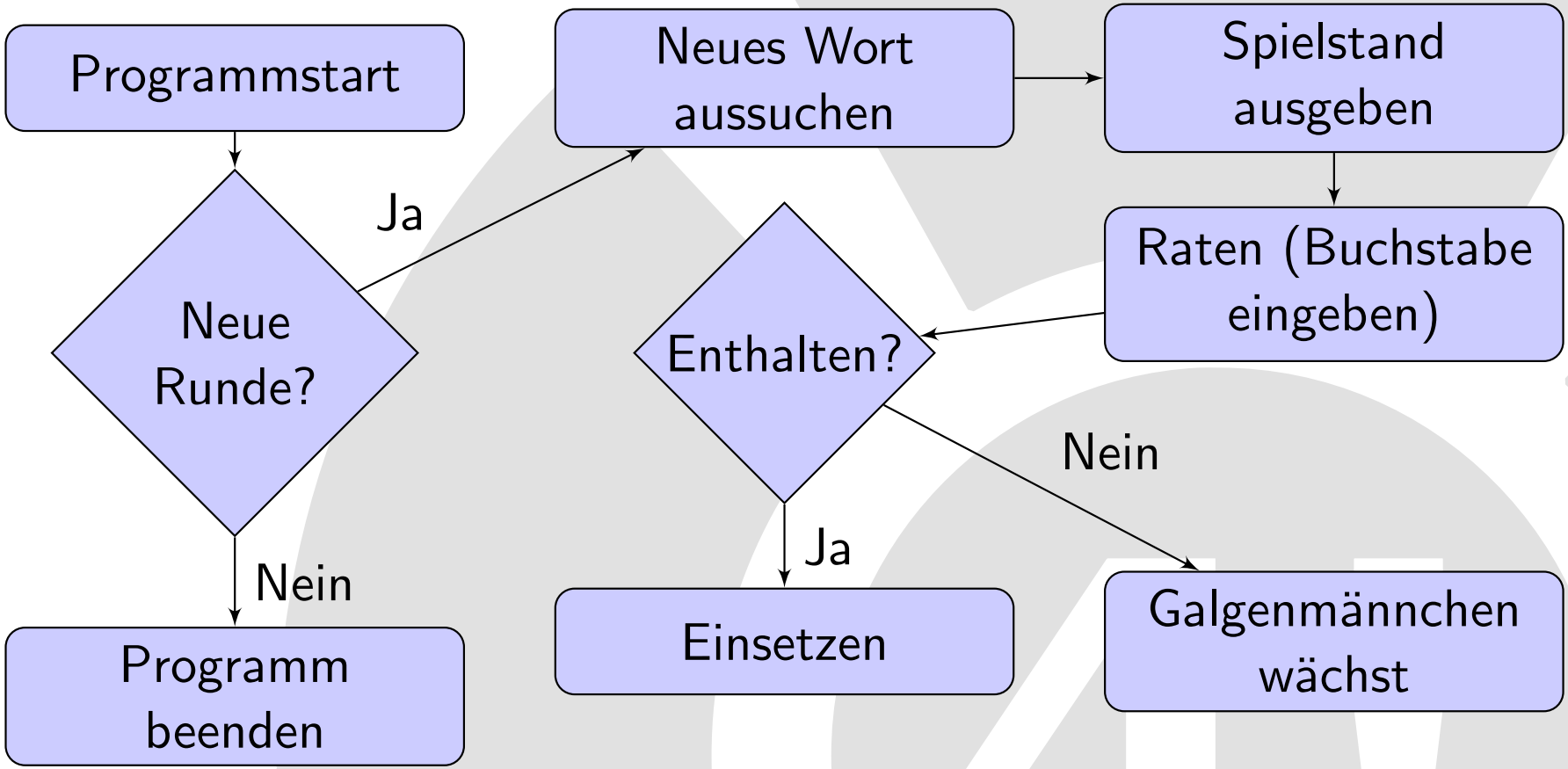
Ablauf aufschreiben



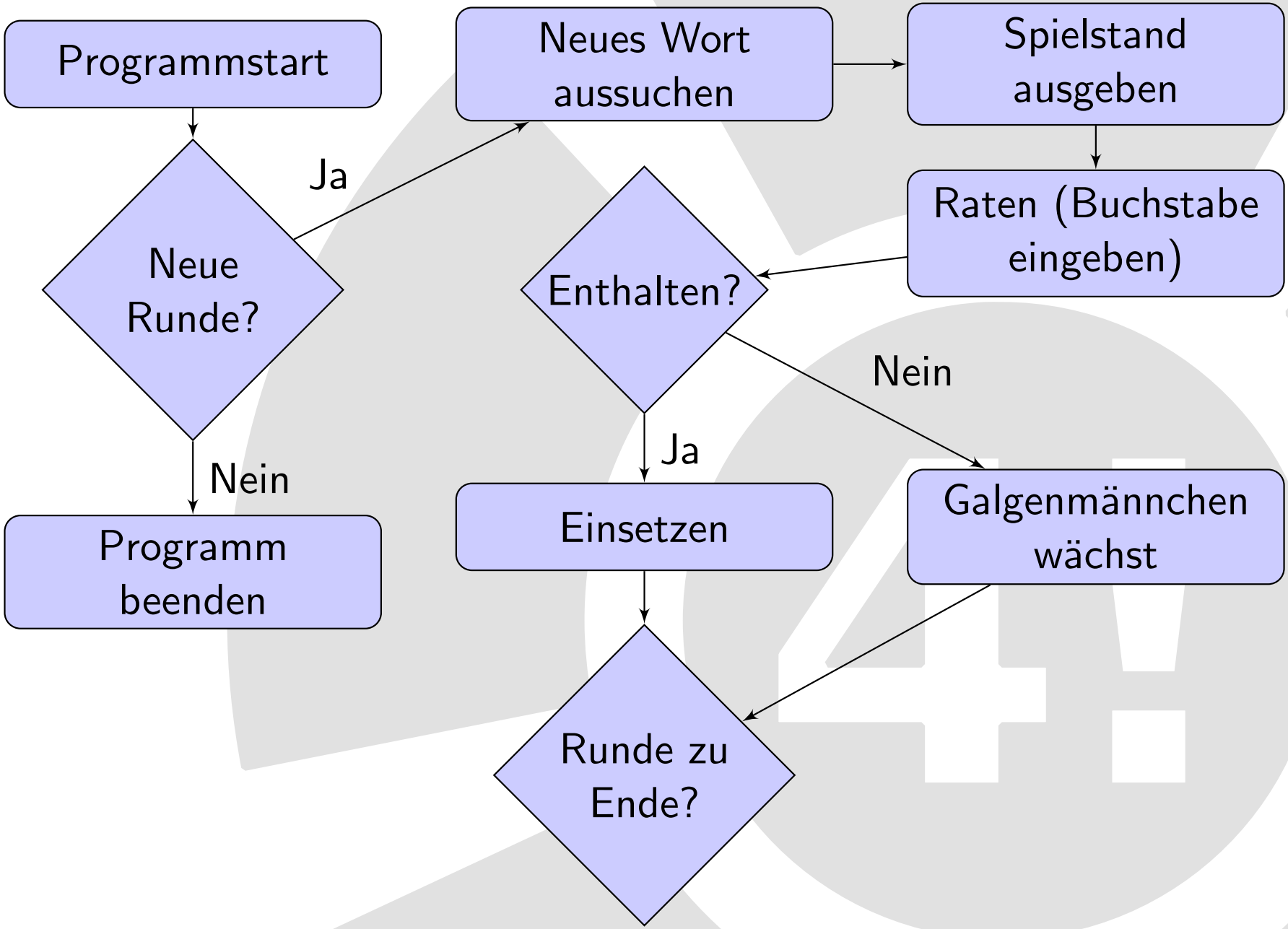
Ablauf aufschreiben



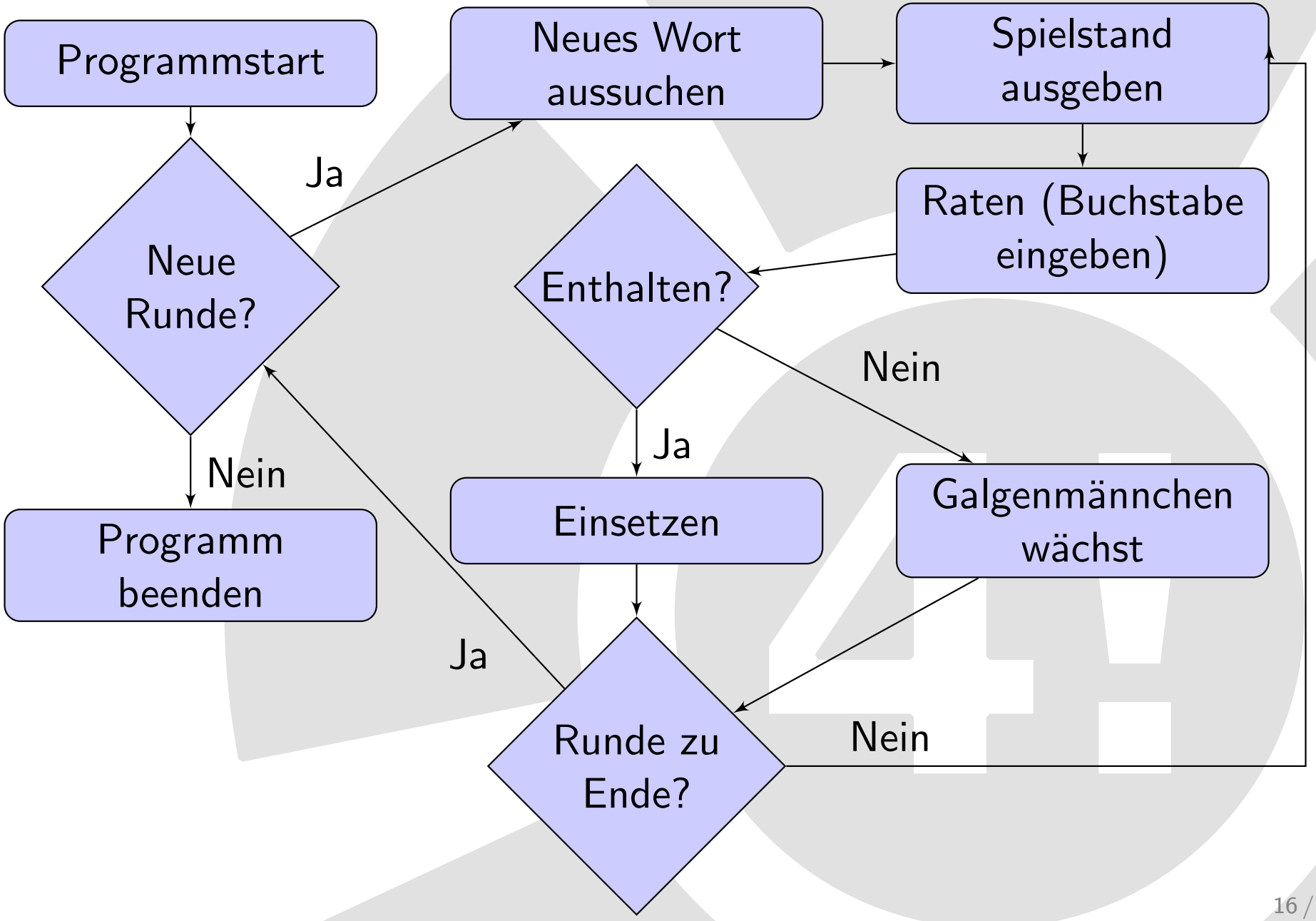
Ablauf aufschreiben



Ablauf aufschreiben



Ablauf aufschreiben



Umsetzung in Methoden

Schritt 3: Struktur in Java umsetzen

→ Erst jetzt fangen wir an, Java-Code zu schreiben!

- Neue Klasse erstellen
- Überlegen, welche Methoden es geben wird
  - Jeder Block aus dem Diagramm wird zu einer Methode
  - Sinnvolle Namen geben!
  - Kann man die Methode noch einmal sinnvoll teilen?

  → Was für Parameter und Rückgabewerte hat die Methode?
  (Muss noch nicht zwingend komplett sein und stimmen)
- `main`-Methode: Ablauf als Kommentare
  - Von dort aus werden später die Methoden aufgerufen

Umsetzung in Methoden

Methoden für Hangman (Blöcke aus dem Ablaufdiagramm)

```
boolean wantNewGame() {}

String chooseNewWord() {}

char enterCharacter() {}

boolean isCharInWord (char character, String secretWord) {}

String insertCharInWord (char character, String secretWord, String
    currentWord) {}

void displayGame () {}

boolean isGameOver() {}
```

- Führt zu Compilerfehler: `missing return statement`
- Wir hätten aber gern etwas, das kompiliert

→ Wir machen *Dummymethoden* draus!

Dummymethoden

Dummymethoden

```
18   boolean wantNewGame() {
19     // TODO: Abfrage, ob neues Spiel gewünscht wird
20     return true;
21   }
22
23   String chooseNewWord() {
24     // TODO: Neues Wort zum Raten ausdenken
25     return "Javakurs";
26   }
27
28   char enterCharacter() {
29     // TODO: Eingabe eines geratenen Buchstabens
30     return 'a';
31   }
```

- Methoden geben erst mal „irgend etwas“ zurück
- Inhaltlich noch nicht richtig, aber zumindest formal

→ Code kompiliert

- TODO-Anzeige für einen selbst, dass noch etwas getan werden muss

Ablauf

Grober Ablauf in `main`-Methode

```
1  public class HangmanStubs {
2    public static void main(String[] args) {
3      // Programmstart
4      // Neue Runde?
5        // Wenn ja, dann suche neues Wort aus
6        // Wenn nein, dann Ende
7      // Neues Wort aussuchen
8      // Spielstand ausgeben
9      // Buchstaben eingeben lassen
10     // Ist Buchstabe im Wort enthalten?
11       // Wenn ja, einsetzen
12       // Wenn nein, Galgen hochzählen
13     // Ist die Runde zu Ende? (Gewonnen oder verloren)
14       // Wenn ja, frage nach neuer Runde
15       // Wenn nein, gehe zurück zur Spielstandausgabe
16   }
```

Methoden implementieren

Schritt 4: Struktur mit Inhalt füllen

- Möglichkeit 1: *Top-Down-Verfahren*
  - bei `main`-Methode anfangen
  - dann zu den Methoden die von `main` aufgerufen werden,
  - dann zu denen, die davon aufgerufen werden
  - usw
- Möglichkeit 2: *Bottom-Up-Verfahren*
  - Mit einzelnen Methoden anfangen, die man für sich testen kann
  - Methoden mit Ausgabefunktion, wo man sofort was sieht
- Auf jeden Fall: Kommentare beachten!
  (Tut mein Code schon das, was im Ablaufplan und in den Kommentaren steht?)
- Auf jeden Fall: Fertige Methoden testen

Inhaltsverzeichnis

Testen: „println-Debugging“

Debugging (deutsch oft: Debuggen)

Finden und Beseitigen der Fehler (Bugs), die ein Programm noch enthält, oft durch schrittweises Durchgehen und Nachvollziehen

Debuggen mit `System.out.println`:
Möglichst überall Werte und Zwischenergebnisse ausgeben

Methode `isCharInWord` fertig programmiert

```
boolean isCharInWord (char character, String secretWord) {
  System.out.println("Suche " + character + " in " + secretWord);
  boolean isInWord = false;
  for (int i = 0; i < secretWord.length() ; i++) {
    if (secretWord.charAt(i) == character) {
      System.out.println("Gefunden an Stelle " + i);
      isInWord = true;
    }
  }
  System.out.println("Ist Buchstabe enthalten: " + isInWord);
  return isInWord;
}
```

Testen: Vorgehen

→ So früh wie möglich,

→ So oft wie möglich,

→ So gründlich wie möglich.

Warum?

Wenn man erst alles schreibt und dann am Schluss testet, ist es schwerer, in 300 Zeilen Code den Fehler zu finden.

Wie?

- Methode mit unterschiedlichen Parametern aufrufen
  - Gültige/zu erwartende Parameter
  - Ungültige/unsinnige Parameter
  - „Randfälle“(0, viel zu hohe Zahl, leerer String...)
- Zwischen- und Endergebnisse ausgeben mit `System.out.println`

Testen: Vorgehen

Beispiele für Testaufrufe

```
/*
 * Gültige Parameter
 */
isCharInWord('a', "Javakurs"); // Sollte true zurückgeben
isCharInWord('b', "Javakurs"); // Sollte false zurückgeben
isCharInWord('=', "Testwort"); // Sollte false zurückgeben

/*
 * Fehlerfälle
 * Umfangreichere Fehlerbehandlung möglich, aber Mittel dafür
 * sind nicht Bestandteil des Javakurs
 */
isCharInWord('a', "");         // Sollte false zurückgeben
isCharInWord('', "Testwort"); // Sollte false zurückgeben
```

Alle Aufrufe und Ergebnisse werden mittels `System.out.println` ausgegeben

→ Prüfen, ob das erwartete Ergebnis auftritt

Ausblick: JUnit

Immer wieder testen ist anstrengend....
Gibt es da keine Hilfsmittel?
$\rightarrow$ JUnit

Beispiele für JUnit

```
import org.junit.*;

public class TestFoobar {

    @Before
    public void setUp() throws Exception {
            //Vorbereitung hier
    }


    @Test
    public void testSomethingElse() {
  assertTrue("a ist enthalten!", isCharInWord('a', "Javakurs") );
    }

}
```

Inhaltsverzeichnis

Einschub: Der Datentyp char

- `char name = 'zeichen';`
- Einzelnes Zeichen, kein Zeichenkette (Das wäre ein String)
- Nicht unbedingt Buchstabe, auch Ziffer, Leerzeichen...
- Steht in einfachen Hochkommata, nicht doppelten
- Man kann damit vergleichen und sogar rechnen
- Großbuchstaben und Kleinbuchstaben sind unterschiedlich

Beispiel für char-Variable

```
char zeichen = 'j';
System.out.println("Eingegebenes Zeichen: " + zeichen);
boolean jGleichJ = (zeichen == 'J'); // ist false
zeichen++;                           // zeichen ist jetzt 'k'
```

Die Java API

API - **A**pplication **P**rogramming **I**nterface

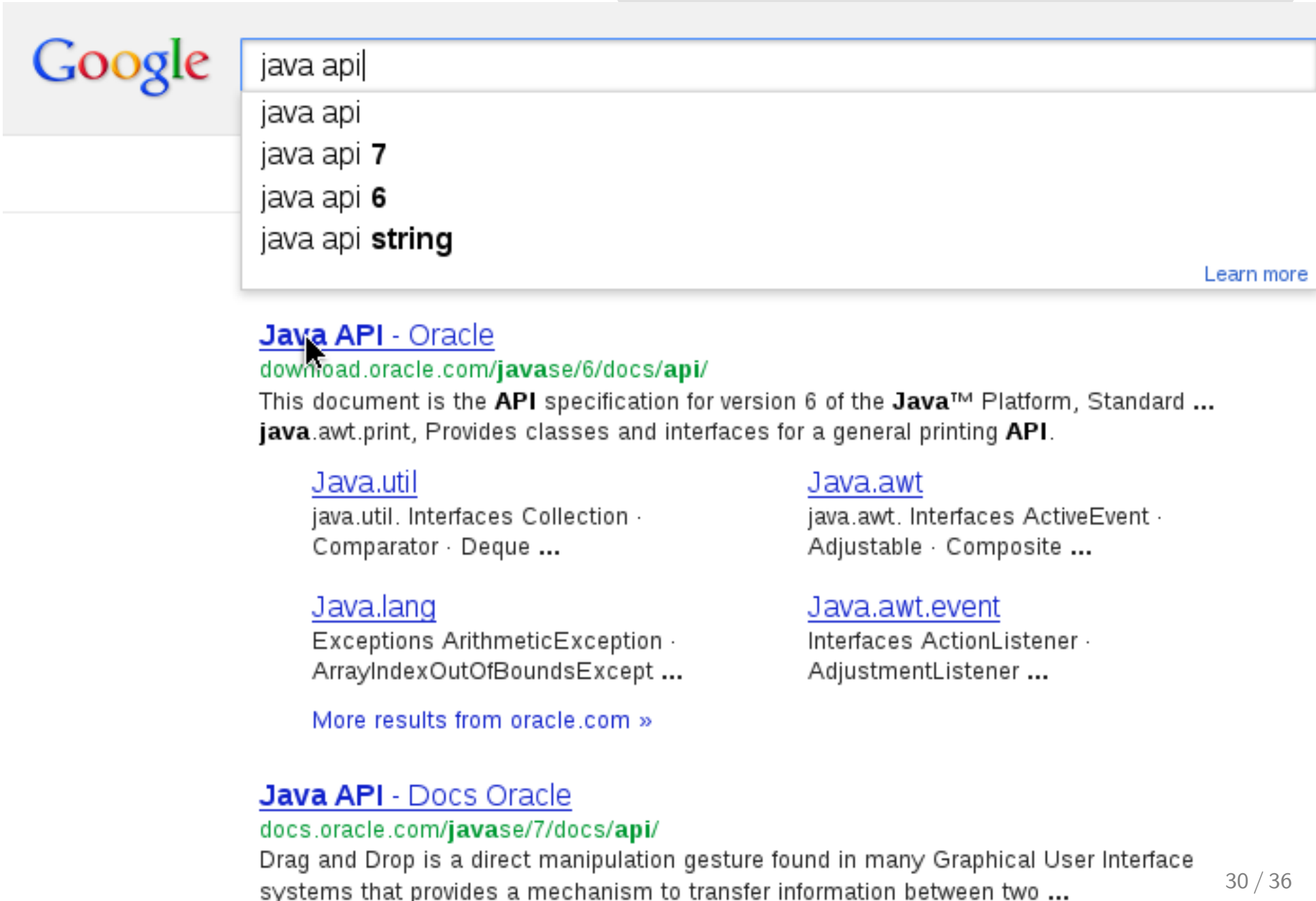
Klassen, Funktionen, Variablen und Datenstrukturen, die zur Verfügung gestellt werden und für eigene Programme benutzt werden können

Java API

Mit „Java API“ meinen wir hier die offizielle Standardbibliothek von Klassen und Methoden, die uns Java zur Verfügung stellt, beziehungsweise deren Dokumentation.

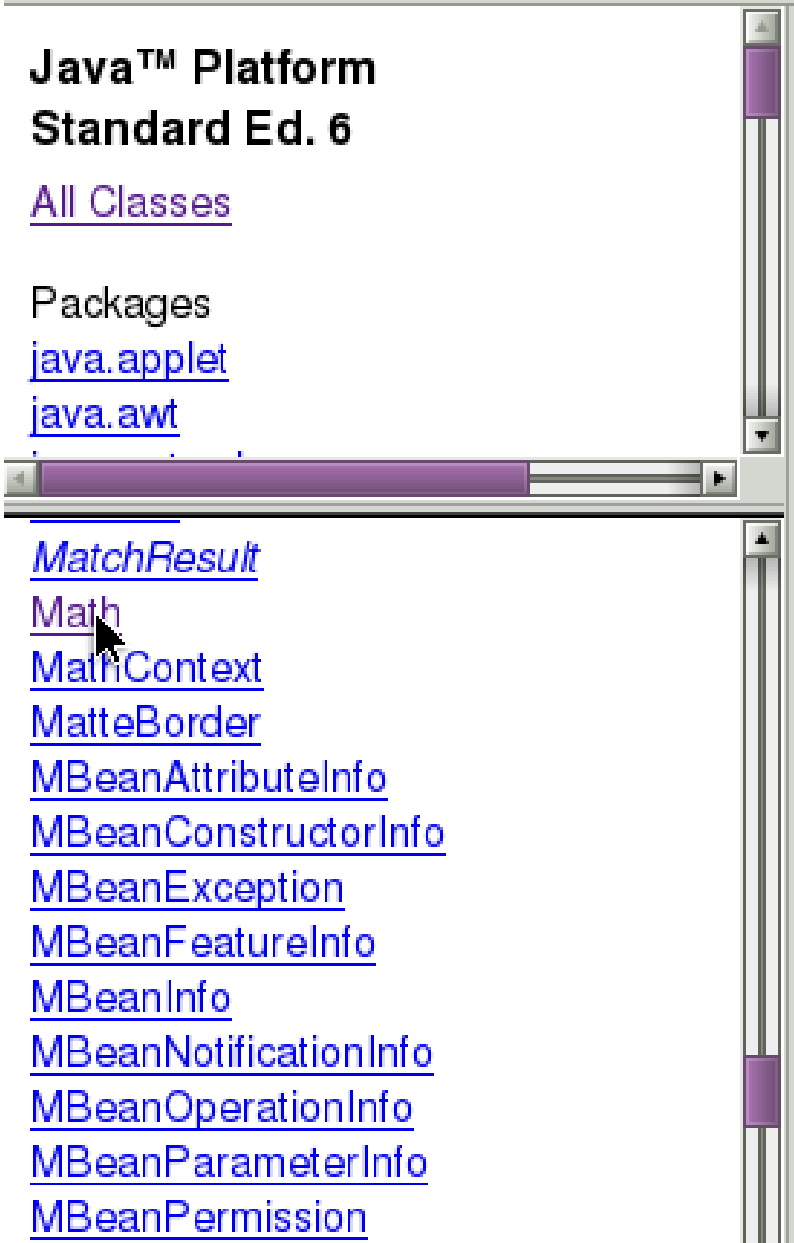
Wenn ihr MPGI1 gehört habt, kennt ihr die **Bibliotheca Opalica**.
Die **Java API** ist sowas ähnliches, nur viel größer und verwirrender.

Java API finden



Etwas in der Java API finden

Im Browser Strg+F, dann z.B. nach Klasse „Math “suchen



Field Summary

static double	E The double value that is closer
static double	PI The double value that is closer

Method Summary

static double	abs(double a) Returns the absolute value of
static float	abs(float a) Returns the absolute value of
static int	abs(int a) Returns the absolute value of
static long	abs(long a) Returns the absolute value of
static double	acos(double a) Returns the arc cosine of a va
static double	asin(double a)

Die Klasse Math

Benutzen der Klasse `Math`

```
1  public class MathExample {
2    public static void main(String[] args) {
3      //Ausgabe aller nutzbaren char-Zeichen (char=16bit)
4      for (int i = 0; i < Math.pow(2,16); i++) {
5          System.out.println(i + " " + (char)i);
6      }
7      //Wurzel von 4 und Cosinus von 2 * PI berechnen & ausgeben
8      double c = Math.sqrt(4);
9      double d = Math.cos(2 * Math.PI);
10     System.out.println("Wurzel: " + c + ", Kosinus: " + d);
11   }
12 }
```

- `Math` enthält Variablen: π = `Math.PI` und e = `Math.E`
  und Methoden: z.B. `sqrt`, `pow`, `abs`, `cos` oder `max`
- Fast alles ist static deklariert: Aufruf mit `Math.`*name*
- Normalerweise sehr effizient/genau implementiert

Die Klasse String

Warum ist das eine Klasse?

- *Per Namenskonvention sind alle groß geschriebenen „Variablentypen“ Klassen.*
- *für alle primitiven Datentypen (int, ...) gibt es auch Wrapper-Klassen (Integer, ...)*

- Wichtig: Strings sind konstant!
  - Konkatenation ist teuer → StringBuilder verwenden
- Nützliche Methoden, z.B. Vergleich, Teilstring liefern, verketten

Beispiel: charAt gibt den Buchstaben an einer bestimmten Position aus

```
String wort = "Javakurs";
System.out.println("3ter Buchstabe (Index 2):" + wort.charAt(2));
```

Beispiel: split trennt einen String nach gegeben Muster (regex) auf

```
String string = "04-12-1989";
String[] parts = string.split("-"); //Trennung am Minus
String part1 = parts[0]; // 04
String part2 = parts[1]; // 12
String part3 = parts[2]; // 1989
```

Collections

Bereits aus MPGI 1 bekannt: Stack, Schlange, Baum
In Java kennt ihr bisher: Array
Es gibt aber viel mehr Datenstrukturen!

- Die Java-API stellt sehr viele Datenstrukturen zur Verfügung
- Die wichtigsten:
  - ArrayList - Array ohne feste Größe
  - ArrayDeque - Stack / Schlange
  - *LinkedList - verkettete Liste, meist langsamer*
  - später wichtig: HashMap, PriorityQueue, ArrayBlockingQueue ...
- Es gibt aber auch Methoden für die Datenstrukturen:
  - sort(), binarySearch(), max(), reverse, ...

Beispiel: Verwendung von sort

```
int[] a = {5,7,1,2,1};
Arrays.sort(a); // [1,1,2,5,7]
```

Zusammenfassung

- Datentyp char: `char name = 'a';` (genau ein Zeichen)
- Lesbarer Code durch:
  - Inline- und Blockkommentare
  - Leerzeichen, Leerzeilen, Einrückung
  - Einfachen statt eleganten Code
  - Sprechende Variablennamen
- Lies Dokumentation, schreib Dokumentation
- Bearbeiten einer komplexen Aufgabe
  1. Aufgabe verstehen
  2. Ablauf aufschreiben
  3. Struktur in Java umsetzen
  4. Struktur mit Inhalt füllen
- Testen durch Debugging mit `System.out.println`
- Java API mit nützlichen Klassen (`Math`, `String`...)
  1. Math
  2. String
  3. Collections

Jetzt: Feedback abgeben!

Bonus: Externe Bibliotheken - JGraphT

Neben der Java-API gibt es viele weitere (freie) Bibliotheken

Für MPGI2 ist besonders JGraphT interessant:

- Enthält alle wichtigen Graphentypen (gewichtet, gerichtet, ..)
- In Kombination mit JGraph ist eine schnelle Visualisierung möglich
- Alle besprochenen Algorithmen sind bereits implementiert
- gut zum Vergleichen der Hausaufgaben
- für nach MPGI sicher zu empfehlen

$\rightarrow$ Wie ruf ich externe Bibliotheken auf?
$\rightarrow$ Wie benutz ich JGraphT?