

EINFACHE STRING- UND TEXTVERARBEITUNG

Repräsentation und Modifikation von Texten in Java

Struktur und Inhalt des Kurses wurden 2012 von Markus Heckner entwickelt.
Im Anschluss haben Alexander Bazo und Christian Wolff Änderungen am
Material vorgenommen. Die aktuellen Folien wurden von Alexander Bazo
erstellt und können unter der [MIT-Lizenz](#) verwendet werden.

AKTUELLER SEMESTERFORTSCHRITT (WOCHE 6)

| Kursabschnitt | Themen | | |
|-------------------------|---|------------------------------|------------------------------|
| Grundlagen | Einführung | Einfache Programme erstellen | Variablen, Klassen & Objekte |
| | Kontrollstrukturen & Methoden | Arrays & komplexe Schleife | |
| Klassenmodellierung | Grundlagen der Klassenmodellierung | | |
| | Vererbung & Sichtbarkeit | | |
| Interaktive Anwendungen | Event-basierten Programmierung | | String- & Textverarbeitung |
| Datenstrukturen | Listen, Maps & die Collections | | Speicherverwaltung |
| | Umgang mit Dateien | | |
| Software Engineering | Planhaftes Vorgehen bei der Softwareentwicklung | | |
| | Qualitätsaspekte von Quellcode | | |

PINGO-QUIZ



<https://pingo.coactum.de/442924> 

DAS PROGRAMM FÜR HEUTE

- Zustände und Optionen durch Enums abbilden
- Repräsentation von Zeichen über den char-Datentyp
- Zeichenketten mit Strings abbilden und modifizieren
- Implementieren einer wiederverwendbaren *Utility*-Klasse für die Textverarbeitung

ENUMERATION

WIE REPRÄSENTIEREN WIR OPTIONEN IM COMPUTER?

1

Umfragedaten

ja

nein

vielleicht

2

Schwierigkeitslevel

- Leicht
- Mittel
- Schwer

3

Zustände



WIND

NO_WIND

Hinweis: Zur internen Verarbeitung (Rechnen, vergleichen, *switchen*, ...) bietet es sich an, diese Werte als Ganzzahlen zu speichern. Jeder Option wird eine Zahl zu geordnet.

BEISPIEL: KONSTANTEN (1/2)

```
1 public static final int YES = 0;
2 public static final int NO = 1;
3 public static final int MAYBE = 2;
4 public static final int I_DONT_KNOW = 3;
```

Wir benutzten Konstanten (int), um jedem Status einen eindeutigen Wert zu zuweisen. Im Code kann der Programmierer die Werte anhand des Namens identifizieren. Intern kann der Computer mit den Zahlen arbeiten. Die Bezeichner erlauben semantischen, gut lesbaren Code:

```
1 if(selectedOption == I_DONT_KNOW) {
2     // do something
3 }
```

BEISPIEL: KONSTANTEN (2/2)

Bei der Verwendung von int-Konstanten, stoßen wir auf ein Problem:
Variablen können mit Werten belegt werden, die nicht vorgesehen sind, z.B.
`selectedOption = 4.` Die Konstanten repräsentieren nur Zahlen. Bis auf den
Namen gibt es keine Möglichkeiten, den Inhalt zu beschreiben:
`System.out.println(selectedOption)` gibt 3 aus.

- Die Abbildung von Zuständen durch Konstanten ist unzureichend vor Fehlersituationen geschützt.
- Die Verwendung von int-Werten macht den Code übersichtlicher und erleichtert das *Verrechnen* von Werten, ist jedoch wenig informativ.

BEISPIEL: ENUMS (1/2)

```
1 public enum Option {  
2     YES,  
3     NO,  
4     MAYBE,  
5     I_DONT_KNOW  
6 }
```

Hinweis: *Enums (Enumeration Types)* stellen einen besonderen Datentyp dar, der das Erstellen eines abgeschlossenen Wertraums für entsprechende Variablen ermöglicht. Mit *Enums* können wir eine Liste an möglichen Optionen erstellen. Der Name des *Enums* wird als Datentyp für die entsprechenden Variablen verwendet. Die einzelnen Werte werden kommasepariert angegeben. Jeder Eintrag hat einen individuellen Ganzahlwert (beginnend bei 0) und einen eindeutigen Namen (String).

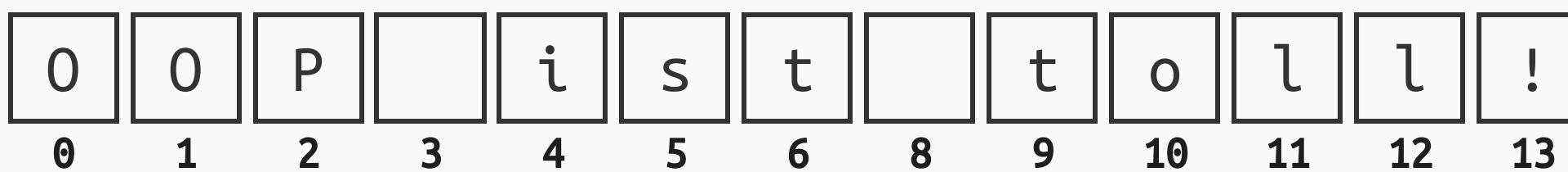
BEISPIEL: ENUMS (2/2)

```
1 // Der Zugriff auf die Werte erfolgt über die Enum- "Klasse"
2 Option selectedOption = Option.MAYBE;
3
4 // Hier wird der Name (MAYBE) der Option als Text ausgegeben
5 System.out.println(selectedOption);
6
7 // Compilerfehler: Der Wert `42` ist kein gültiger Eintrag für die Enum-Variable
8 selectedOption = 42;
```

EINZELNE ZEICHEN: CHAR UND ASCII

CHARS

Der primitive Datentyp `char` dient in Java dem Speichern eines einzelnes Zeichen. Die Klasse `Character` bündelt Methoden, mit denen diese Zeichen verarbeitet werden können. Ein `String` besteht intern (*Information Hiding*) aus einem *Array* von `char`.



Jedes Zeichen des Textes `00P is toll!` ist `toll!`, auch die Leer- und Satzzeichen, werden durch einzelnen `char`-Werten, die in der `String`-Instanz in der korrekten Reihenfolge in einem Array abgespeichert werden.

INTERNE REPRÄSENTATION VON ZEICHEN

- Variablen vom Typ char werden im Computer mit unterschiedlichen Zahlwerten repräsentiert.
- Jedes Zeichen hat eine eindeutige Nummer, die für die Verarbeitung verwendet wird.

Welche Nummern sind welchen Zeichen zugeordnet?

ASCII & UNICODE

Java unterstützt den **Unicode-Standard**. Zur internen Repräsentation von Zeichen wird *UTF-16* verwendet. ASCII ist eine Teilmenge dieses Standards und kann für uns zur leichten Identifikation von den wichtigsten Zeichen verwendet werden.

VORTEILE DER ASCII-REPRÄSENTATION

- Groß- und Kleinbuchstaben sind jeweils alphabetisch und in aufsteigender Sortierung in der Tabelle eingetragen.
- Auch Ziffern sind ihrem numerischen Wert nach sortiert.
- Wir können bei Berechnungen und Sortierungen also die numerischen Werte der char-Zeichen verwenden!

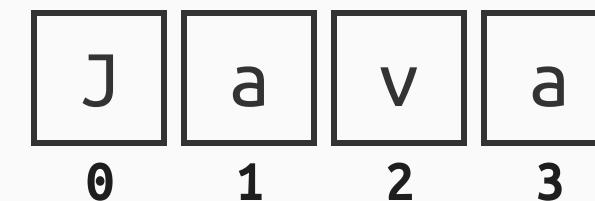
```
1 // A => 65; B => 66
2 char c1 = 'A'; char c2 = 'B';
3 if(c1 < c2) {
4     // the char in c2 is alphabetically ordered before the char c1
5 }
```

CHARS AUS STRINGS EXTRAHIEREN

Strings sind Ketten einzelner Zeichen. Ein einzelnen Zeichen kann über seine Position innerhalb eines Strings abgefragt werden:

```
1 Scanner in = new Scanner(System.in);  
2 String input = in.nextLine();
```

Eingegeben wird der Text *Java*. Intern wird der Text in der Variable *input* als *Array* abgebildet.

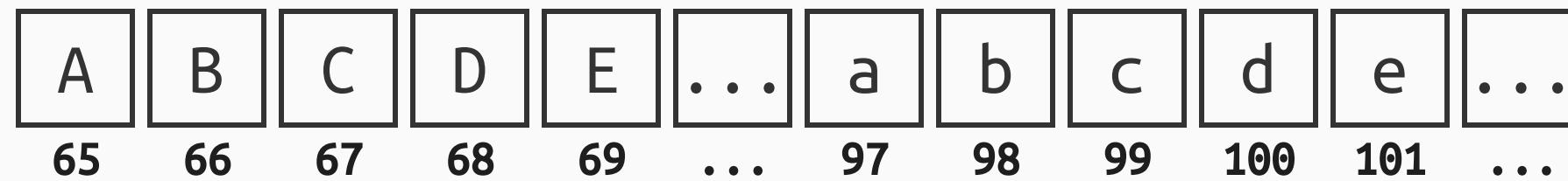


Wir können einzelnen Zeichen auslesen:

```
1 char fistChar = input.charAt(0);
```

ZÄHLEN UND RECHNEN MIT CHARS

Durch die interne Repräsentation als Zahlwert ist es möglich, implizit oder explizit, zwischen `char` und `int` zu *casten*.



ZÄHLEN

```
1 for(char ch = 'A'; ch < 'Z'; ch++) {  
2     System.out.println(ch);  
3 }
```

RECHNEN

```
1 char ch = 'B';  
2 int positionInAlphabet = ch - 'A';  
3 char lowerCase = (char) ('a' + positionInAlphabet);
```

CHAR VS CHARACTER

- `char` ist ein primitiver Datentyp und verhält sich so wie die anderen bekannten primitiven Datentypen (`int`, `boolean`, `double`, ...).
- `char` ist keine Klasse! Also gibt es auch kein Objekte (Instanzen) oder Methoden!
- **Aber:** `Character` ist eine *Utility-Klasse*, die statische Methoden für die Zeichen-Verarbeitung bietet, z.B. `Character.toUpperCase(char ch)`.

NÜTZLICHE METHODEN DER CHARACTER-KLASSE

| Methode | Beschreibung (aus JDK-Dokumentation) |
|--|--|
| static boolean isDigit(char ch) | <i>Determines if the specified character is a digit.</i> |
| static boolean isLetter(char ch) | <i>Determines if the specified character is a letter.</i> |
| static boolean isLetterOrDigit(char ch) | <i>Determines if the specified character is a letter or digit.</i> |
| static boolean isLowerCase(char ch) | <i>Determines if the specified character is a lowercase character. (gibt auch für uppercase)</i> |
| static boolean isWhitespace(char ch) | <i>Determines if the specified character is white space according to Java</i> |
| static char toLowerCase(char ch) | <i>Converts the character argument to lowercase ... (gibt auch für uppercase)</i> |

TEXTE VERARBEITEN MIT DER STRING-KLASSE

GRUNDSÄTZLICHES ZUR STRING-KLASSE

String ist eine Klasse, von der Instanzen erzeugt werden. Methoden der Klasse werden in der Regel auf den Instanzen aufgerufen:

```
1 String input = "Java ist toll";
2 input = input.toUpperCase();
```

Strings sind *immutable*. D.h. nach der Initialisierung kann der Inhalt nicht mehr geändert werden. Alle Methoden arbeiten daher nach dem gleichen Prinzip: String als **Input**, neuer String als **Output**. Auch die Verkettung mehrere *Strings* mit dem Operator + erzeugt immer einen neuen *String* aus den verwendeten Bestandteilen!

STRING VERGLEICHEN (1/2)

Was überprüfen wird beim Vergleich zweier Objekt-Variablen mithilfe des `==`-Operators? Wir prüfen, ob die Variablen auf die selben Objekte zeigen (Objekte verhalten sich anders keine primitive Daten)!

```
1 String value1 = new String("Java ist toll");
2 String value2 = new String("Java ist toll");
3
4 System.out.println(value1 == value2); //false
5 value2 = value1;
6 System.out.println(value1 == value2); //true
```

Hinweis: Im Beispiel werden explizit neue String-Objekte erzeugt. Wenn Sie *Literale* (ohne new-Operator) verwenden, optimiert der *Compiler* den Code und verwendet für gleiche Literale das gleiche Objekt!

STRING VERGLEICHEN (2/2)

Wie vergleichen wir *Strings* auf inhaltlicher Ebene? Wir verwenden die Instanzmethode `equals`, die die String-Klasse bereitstellt:

```
1 String value1 = new String("Java ist toll");
2 String value2 = "Java " + "ist" + " toll";
3 System.out.println(value1.equals(value2)); // true
```

Hinweis: In der Regel möchten Sie *immer* den Inhalt von *Strings* vergleichen und nicht die jeweiligen Objekte. Verwenden Sie daher grundsätzlich die `equals`-Methode und nutzten Sie den Vergleichsoperator (`==`) nur, wenn Sie dies wirklich müssen.

NÜTZLICHE METHODEN DER STRING-KLASSE

| Methode | Beschreibung (aus JDK-Dokumentation) |
|---|--|
| <code>int length()</code> | <i>Returns the length of this string.</i> |
| <code>char charAt(int index)</code> | <i>Returns the char value at the specified index.</i> |
| <code>String substring(int beginIndex, int endIndex)</code> | <i>Returns a new string that is a substring of this string (from beginIndex to endIndex).</i> |
| <code>String substring(int beginIndex)</code> | <i>Returns a new string that is a substring of this string (from beginIndex to end).</i> |
| <code>boolean equals(String s2)</code> | <i>Compares this string to the specified String.</i> |
| <code>int compareTo(String s2)</code> | <i>Compares two strings lexicographically. 0 if strings are equal, -1 for "smaller", 1 for "larger" String</i> |
| <code>int indexOf(char c)</code> | <i>Returns the index of the first occurrence of the character, or -1 if not found.</i> |
| <code>int indexOf(String s)</code> | <i>Returns the index of the first occurrence of the</i> |

FUN WITH STRINGS

UNSERE AUFGABE

Wir schreiben eine Klasse `StringHelper`, in der mehrere *Utility*-Funktionen für die Arbeit mit *Strings* zusammengefasst werden:

- Zählen von Groß- bzw. Kleinbuchstaben
- Ersetzen von Substrings
- Umdrehen von Strings
- Palindrom-Check
- Zerlegen des Strings in Einzeltoken

DIE KLASSE STRINGHELPER

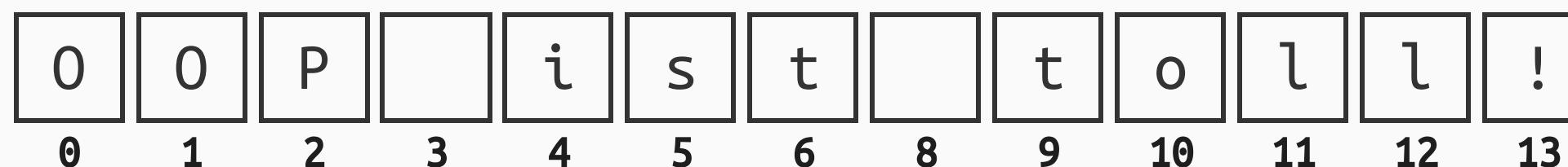
Die Klasse arbeitet nach folgendem Prinzip: Bei Instanziierung wird eine String-Instanz übergeben (Vgl. *Composition*). Alle Methoden arbeiten intern mit diesem String:

```
1 // Einlesen eines Strings über die Kommandozeile
2 String input = getInputFromUser();
3 StringHelper helper = new StringHelper(input);
4
5 String output = helper.reverseString();
```

Einzelne Features der Klasse werden über öffentliche Methoden zugänglich gemacht. Wir arbeiten dabei nach dem Prinzip des *Information Hidings*: *Clients* nutzen eine öffentliche Schnittstelle, interne Prozess werden durch private Methoden abgebildet. Wenn möglich, nutzen wir bereits implementierte Funktionen für die Realisierung komplexerer Aufgaben.

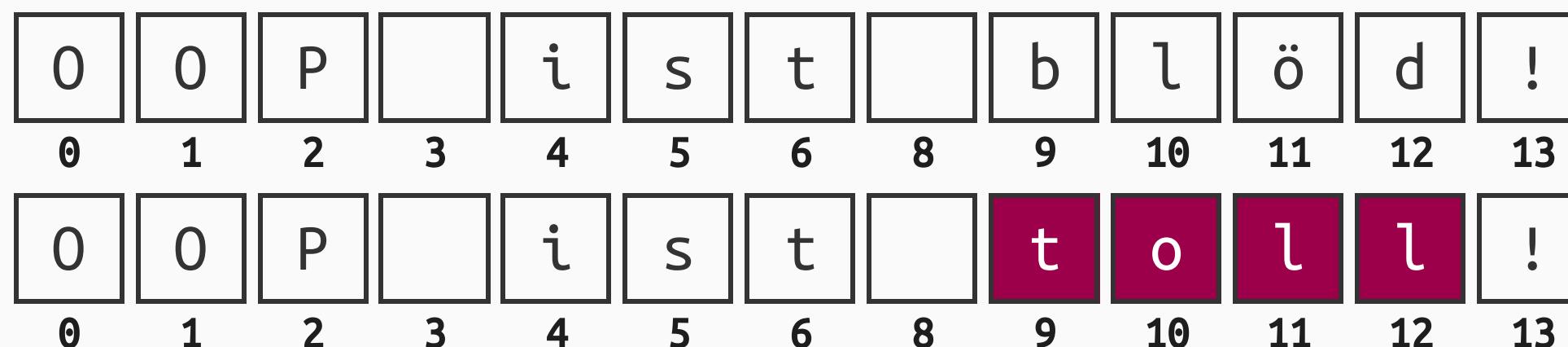
ZÄHLEN VON GROSS- UND KLEINBUCHSTABEN

- Was ist unser Ziel?
- Was sind die Parameter und was der Rückgabewert?
- Was ist das eigentliche Problem?
- Wie kann der Algorithmus beschrieben werden?
- Brauchen wir Hilfsmethoden (Dekomposition)?



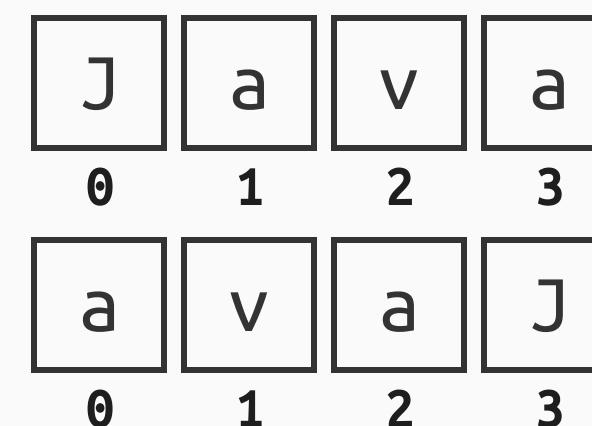
ERSETZEN VON EINEM SUBSTRING

- Was ist unser Ziel?
- Was sind die Parameter und was der Rückgabewert?
- Was ist das eigentliche Problem?
- Wie kann der Algorithmus beschrieben werden?
- Brauchen wir Hilfsmethoden (Dekomposition)?



UMDREHEN DES STRINGS

- Was ist unser Ziel?
- Was sind die Parameter und was der Rückgabewert?
- Was ist das eigentliche Problem?
- Wie kann der Algorithmus beschrieben werden?
- Brauchen wir Hilfsmethoden (Dekomposition)?



PALINDROM-CHECK

- Was ist unser Ziel?
- Was sind die Parameter und was der Rückgabewert?
- Was ist das eigentliche Problem?
- Wie kann der Algorithmus beschrieben werden?
- Brauchen wir Hilfsmethoden (Dekomposition)?

STRINGS ZERLEGEN

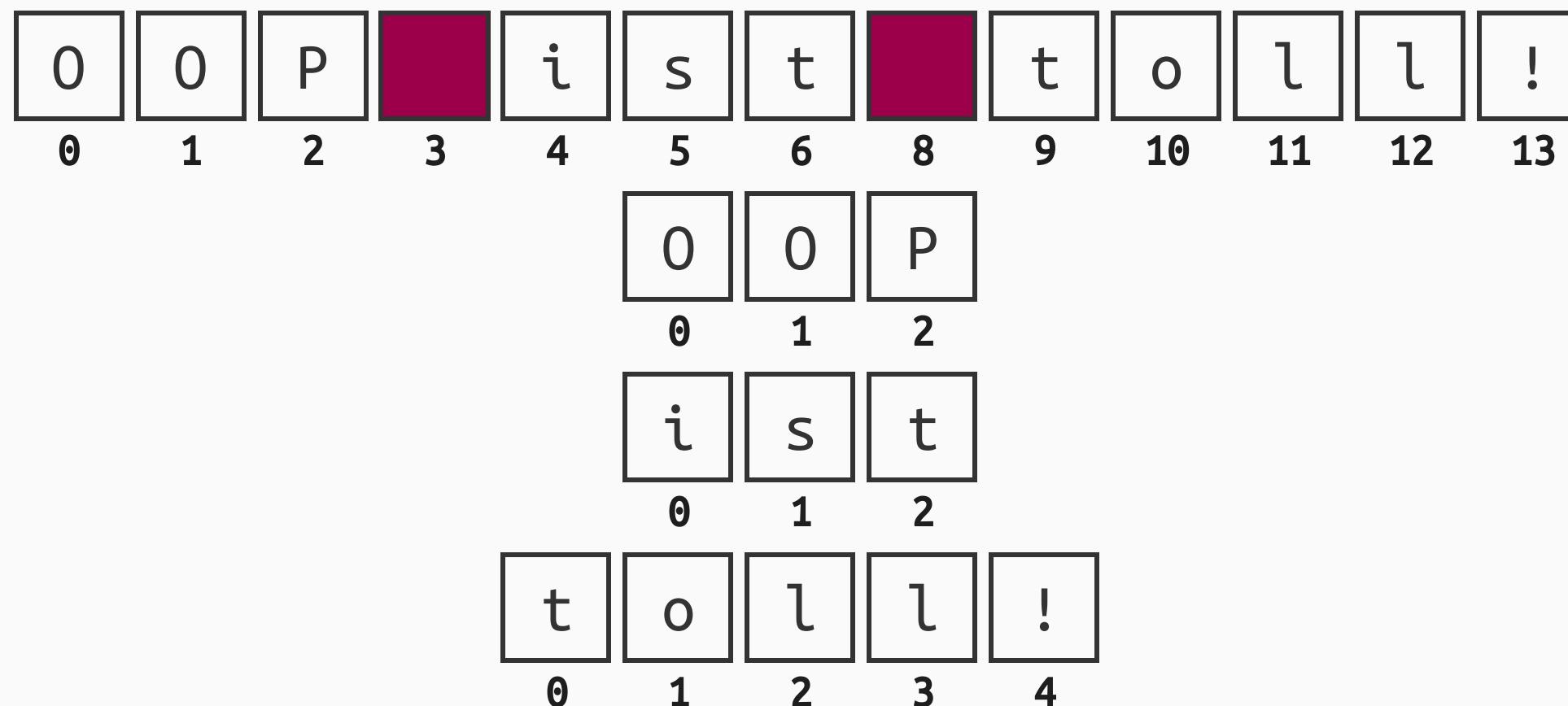
Mit Hilfe der Instanzmethode `split` der `String`-Klasse können Sie eine Zeichenkette entlang beliebiger Zeichen auftrennen und dadurch Tokens erzeugen. Als Ergebnis erhalten Sie ein Array mit allen Einzeltokens, die Zeichen, die als Trennstelle verwendet werden, sind im Ergebnis-Array nicht mehr vorhanden:

```
1 String input = "this_is_a_string";
2 String[] tokens = input.split("_");
3 // result: ["this", "is", "a", "string"]
```

Hinweis: Das JDK bietet für komplexere Zerlegungsaufgaben die Klasse `StringTokenizer` an. Auch mit der `Scanner`-Klasse lassen sich ähnliche Aufgaben erledigen. Die Auswahl sollte immer dem grundsätzlichen Problem angepasst werden - für die Aufgaben in diesem Kurs ist die `split`-Methode in der Regel ausreichend.

ZERLEGEN DES STRINGS IN EINZELTOKEN

- Was ist unser Ziel?
- Was sind die Parameter und was der Rückgabewert?
- Was ist das eigentliche Problem?
- Wie kann der Algorithmus beschrieben werden?
- Brauchen wir Hilfsmethoden (Dekomposition)?



KRYPTOGRAPHIE: CAESAR'S CIPHER

Mit unserem neuen Wissen über `char` und `String` können wir versuchen, Texte zu verschlüsseln. Eine der einfachsten und bekanntesten Methoden dazu ist der *Caesar Code*. Jeder Buchstabe des zu verschlüsselnden Text wird durch einen anderen ersetzt, der den im Alphabet den Abstand n vom ursprünglichen Buchstaben hat. Wenn Sie n kennen, können Sie den Text wieder entschlüsseln.

ZUSAMMENFASSUNG

- Mittels Enumeration werden einer zusammengehörigen Set von Werten lesbare Namen gegeben – Der Programmierer greift anhand von benannten Konstanten auf die Werte zu
- Einzelne Zeichen werden in Java als char gespeichert – Jedem char ist eine Ganzzahl zugeordnet
- Strings sind *immutable*, d.h. nach Erzeugung eines Objekts nicht mehr veränderbar
- Strings lassen sich mit der Methode split in einzelne Tokens (*Substrings*) zerlegen

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT. WENN SIE MÖCHTEN, SEHEN WIR UNS IM ANSCHLUSS IN DER ZENTRALÜBUNG!

Fragen oder Probleme? In allgemeinen Angelegenheiten und bei Fragen zur Vorlesung wenden Sie sich bitte an Alexander Bazo (alexander.bazo@ur.de). Bei organisatorischen Fragen zu den Studienleistungen und Übungsgruppen schreiben Sie bitte Florin Schwappach (florin.schwappach@ur.de). Bei inhaltlichen Fragen zu den Übungen, Beispielen und Studienleistungen schreiben Sie uns unter mioop@mailman.uni-regensburg.de.

QUELLEN

Eric S. Roberts, *The art and science of Java: an introduction to computer science, New international Edition*, 1. Ausgabe, Pearson, Harlow, UK, 2014.