Universität Bremen

Praktische Informatik 1 Sammlungen mit fester Größe – Arrays

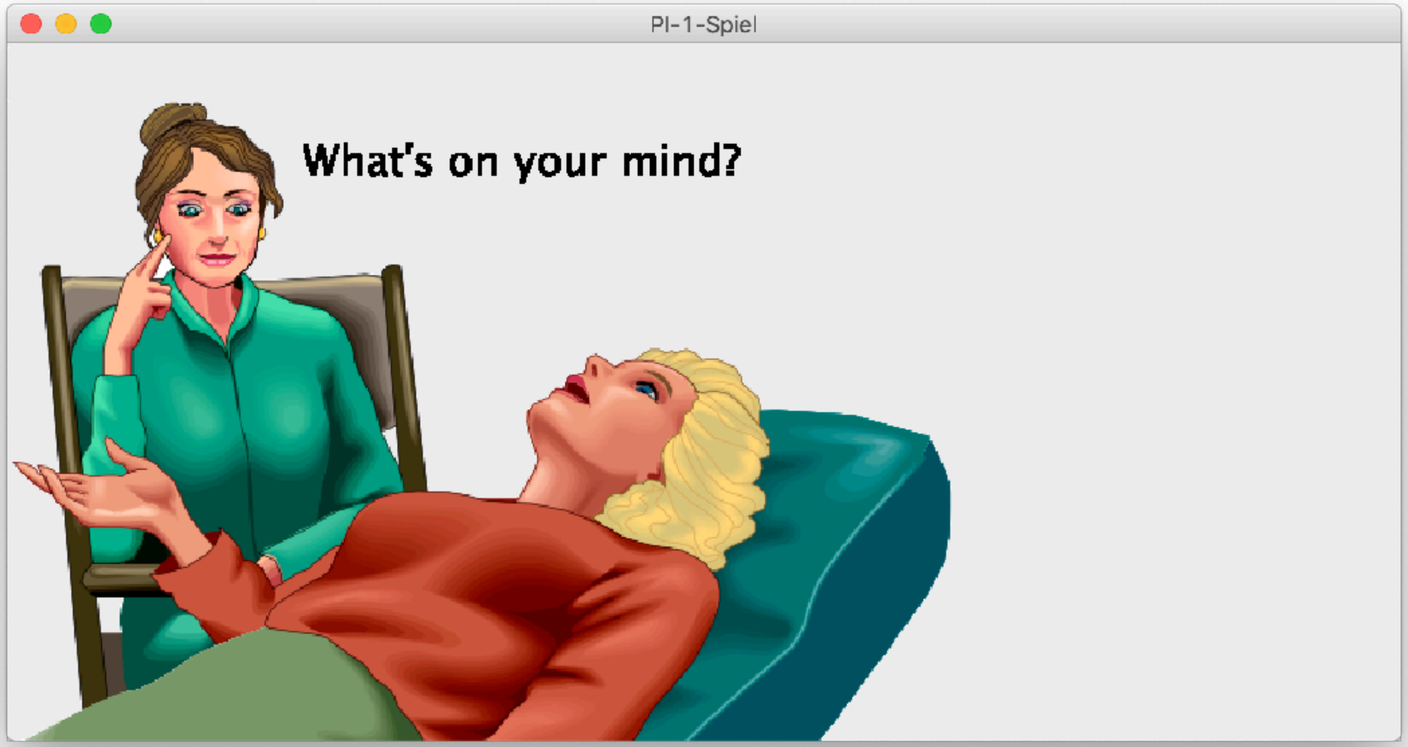
Thomas Röfer

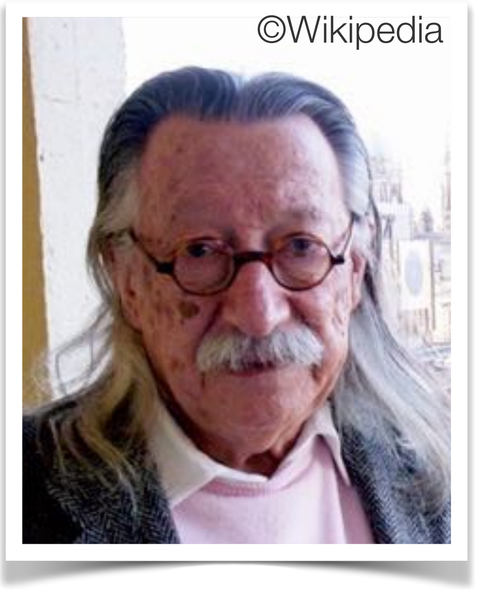
Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

ELIZA: Demo 1

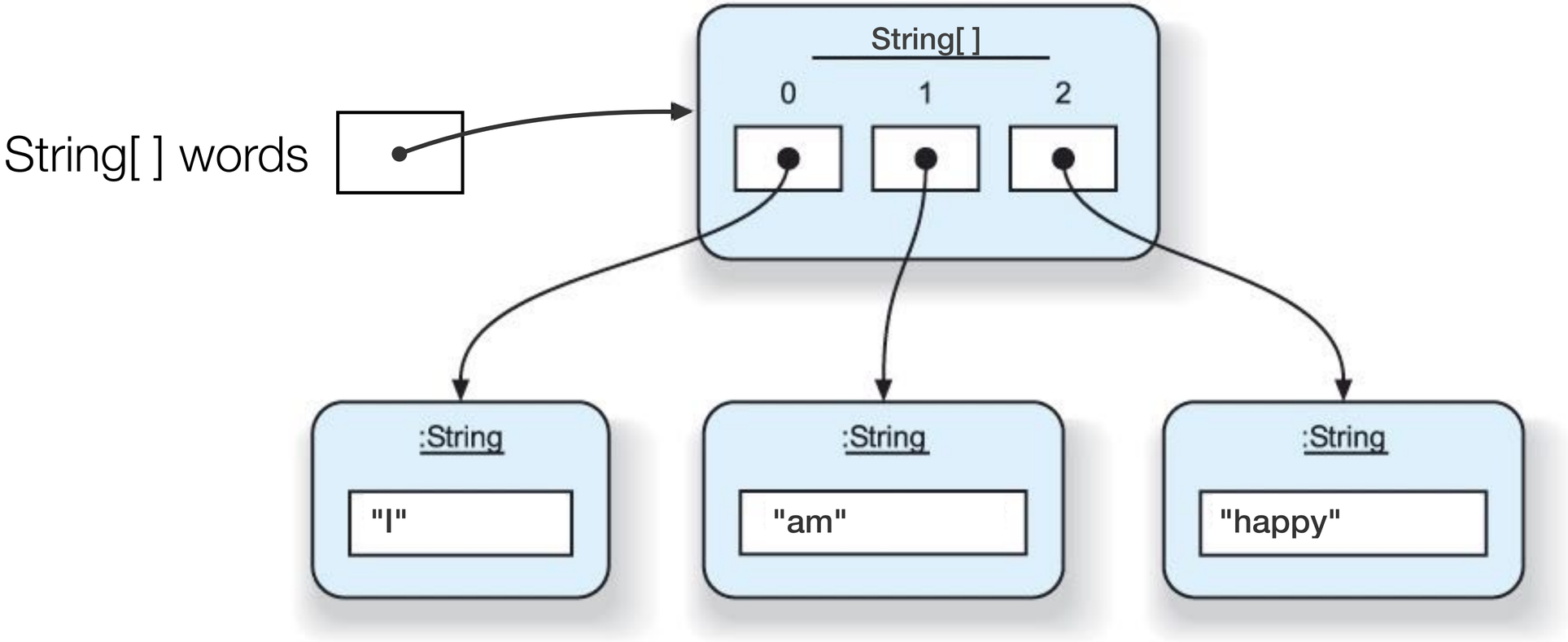




Joseph Weizenbaum
1923-2008

Joseph Weizenbaum: ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine. in: Communications of the ACM. New York 9.1966,1. ISSN 0001-0782

Sammlungen fester Größe (Arrays)



Universität Bremen

Sammlungen fester Größe (Arrays)

- Ein **Array** ist ein **Objekt**, das eine **feste Anzahl** von Elementen gleichen Typs speichert, auf die mit einem **Index** zugegriffen wird
- Kann **Objekte** und **primitive Datentypen** speichern
- Deklaration
- Konstruktion
- Initialisierung
- Zugriff

```
int[ ] a;
```

```
a = new int[6];
```

```
a = new int[ ] {4, 8, 15, 16, 23, 42};
```

```
int[ ] a = {4, 8, 15, 16, 23, 42};
```

```
a[4] = a[1] + a[2];
```

Universität Bremen

Zweidimensionale Arrays

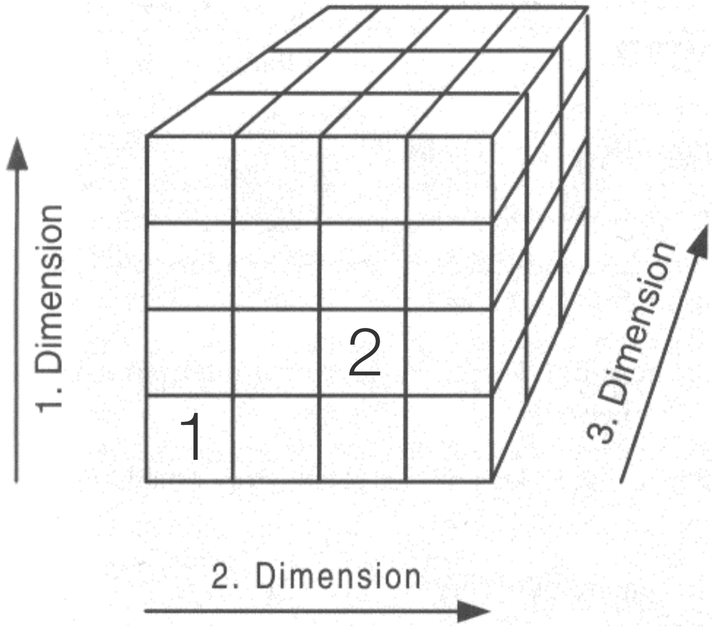
```
final String[ ][ ] replacements = {
    {"I", "you"},
    {"me", "you"},
    {"am", "are"},
    {"my", "your"}
};
```

```
final String[ ][ ] replacements = new String[4][2];
replacements[0][0] = "I";
replacements[0][1] = "you";
replacements[1][0] = "me";
                :
```

	0	1
0	"I"	"you"
1	"me"	"you"
2	"am"	"are"
3	"my"	"your"

Mehrdimensionale Arrays

- Eine beliebige Anzahl von Dimensionen ist möglich
- Speicherplatzverbrauch bedenken!




```
int[ ][ ][ ] a = new int[4][4][3];
a[0][0][0] = 1;
a[1][2][0] = 2;
```

Universität Bremen

Mehrdimensionale Arrays

- Ein zweidimensionales Array ist ein Array von Arrays
- Die Länge eines Arrays steht in seinem konstanten Attribut **length**
- Arrays können in weiteren Dimensionen unterschiedliche Größen haben

	0	1
0	"I"	"you"
1	"me"	"you"
2	"am"	"are"
3	"my"	"your"

	0	1	2	3
0	0	1	2	
1	3	4		
2	5	6	7	8

```
int[ ][ ] a = {
    {0, 1, 2},
    {3, 4},
    {5, 6, 7, 8},
};
int b = a.length; // 3
int c = a[2].length; // 4
```

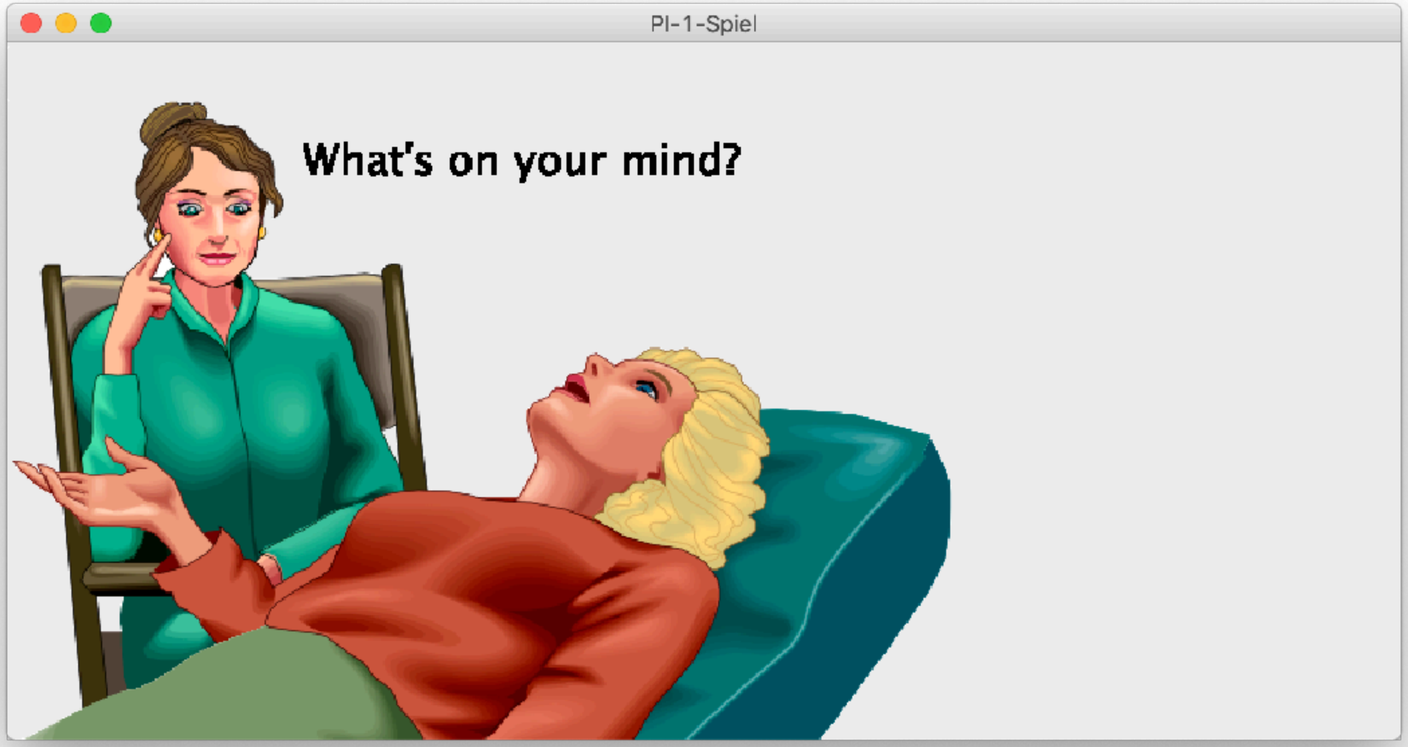
Universität Bremen

Vergleich Array, ArrayList und String

	Anzahl Elemente	Lesen	Schreiben
***Typ*[ ] var**	var**.length**	var**[***index***]**	var**[***index***]** = *wert*;
ArrayList<*Typ*> var	var**.size()**	var**.get(***index***)**	var**.set(***index***, *wert*)**;
String var	var**.length()**	var**.charAt(***index***)**	*nicht möglich*

- **charAt** liefert einen Wert vom Typ **char** (UTF-16-Zeichen)
  - Literale: **'***zeichen***'**, z.B. **'A'**, **'a'**, **'0'**, **'␣'**, **'\n'**, **'\t'**, **'\u0020'**, **0** ... **65535**

ELIZA: Demo 2



Universität Bremen

Hinzurechnen

- Java erlaubt das Verbinden einer Rechenoperation mit der Zuweisung

```
i += 17;
a[a[1]] %= 4;
x *= x;
```

- L-Wert **op=** R-Wert entspricht
  L-Wert **=** L-Wert **op** R-Wert

- Der L-Wert wird aber nur einmal ausgewertet

  - **a += 7** entspricht **a = a + 7**

  - **b[a += 1] += 7** entspricht aber nicht **b[a += 1] = b[a += 1] + 7**

Universität Bremen

Inkrement/Dekrement

- **Pre-Inkrement/Dekrement**: Erhöht/verringert den Wert der Variablen und liefert **diesen** als Ergebnis zurück
  - **++i** entspricht **(i += 1)**
- **Post-Inkrement/Dekrement**: Erhöht/verringert den Wert der Variablen und liefert den **alten Wert** als Ergebnis zurück
  - **i--** entspricht **((i -= 1) + 1)**

```
int i = 0;
a[++i] = 17; // a[1] = 17
a[++i] = 18; // a[2] = 18
--a[i]; // a[2] = 17;
```

```
int i = 0;
a[i++] = 17; // a[0] = 17
a[i++] = 18; // a[1] = 18
a[i]--; // a[2] -= 1;
```

Universität Bremen

Vorrang von Operatoren

- **Bindungskraft** (**Präzedenz**): Was wird zuerst ausgewertet, wenn verschiedene Operatoren in einem Ausdruck verwendet werden?
  - z.B. Punkt- vor Strichrechnung: **1 + 2 * 3 = 1 + (2 * 3) = 7**
- **Assoziativität**: Was wird zuerst ausgewertet, wenn Operatoren gleicher Bindungsstärke in einem Ausdruck verwendet werden?
  - z.B. links bei Minus: **3 - 2 - 1 = (3 - 2) - 1 = 0**
- Die Auswertungsreihenfolge kann manuell durch **Klammerung** geändert werden
  - z.B. **(1 + 2) * 3 = 9**

Universität Bremen

Vorrang von Operatoren

Bindungskraft

Operator	Assoziativität	Beschreibung
., (), [ ]	links	Objektzugriff, Funktionsaufruf und Array-Index
++, --	links	Post-Inkrement/Dekrement
-, +, !, ~, ++, --	rechts	Vorzeichen, logisches Nicht, arithmetisches Nicht, Pre-Inkrement/Dekrement
new, **(***Typ***)**	rechts	Objekterzeugung und Typumwandlung
***, /, %**	links	Multiplikation, Division, Modulo
+, -	links	Addition und Subtraktion
<<, >>, >>>	links	Links- und Rechtsverschiebung
<, <=, >, >=, instanceof	links	Kleiner, Kleiner-Gleich, Größer, Größer-Gleich, Typtest
==, !=	links	Gleichheit und Ungleichheit
&	links	arithmetisches Und
^	links	arithmetisches Exklusiv-Oder
|	links	arithmetisches Oder
&&	links	logisches Und
||	links	logisches Oder
? :	links	Bedingungsoperator
=, +=, -=, *=, /=, %=, ^=, &=, |=, <<=, >>=, >>>=	rechts	Zuweisung und Hinzurechnen
->	rechts	Lambda-Ausdrucks-Pfeil

Universität Bremen

Zählschleife

```
for (int i = start; i < words.length; ++i) {
    sentence += " " + getReplacement(words[i]);
}
```

- Fasst **Initialisierung** einer **Laufvariablen**, **Testen** der **Abbruchbedingung**, Ausführen des Schleifenrumpfs und **Weiterzählen** in einer Anweisung zusammen (in dieser Ausführungsfolge)
- Werden im Initialisierungsteil Variablen deklariert, endet ihre Lebenszeit nach dem Ende der Schleife
- Kann sehr flexibel eingesetzt werden
- Alle Angaben sind optional und es kann mehrere Initialisierungen und Aktualisierungen geben (durch Kommas getrennt)

```
for ( initialisierungen ;
      bedingung ;
      aktualisierungen ) {
    anweisungen
}
```

Analogien zwischen Schleifen: while/do-while

```
while ( bedingung ) {
    anweisungen
}
```

$\triangleq$

```
if ( bedingung ) {
    do {
        anweisungen
    } while ( bedingung );
}
```

```
do {
    anweisungen
} while ( bedingung );
```

$\triangleq^{*}$

```
anweisungen
while ( bedingung ) {
    anweisungen
}
```

$\triangleq^{*}$

```
boolean first = true;
while (first || ( bedingung )) {
    first = false;
    anweisungen
}
```

* Unter der Annahme, dass *anweisungen* kein **continue** und kein **break** enthält

Analogien zwischen Schleifen: for/while

```
for ( initialisierungen; bedingung; aktualisierungen ) {
    anweisungen
}
```

$\triangleq^{*}$

```
{ // Manchmal ',' → ';'
    initialisierungen;
    while ( bedingung ) {
        anweisungen
        aktualisierungen;
    }
}
```

* Unter der Annahme, dass *anweisungen* kein **continue** enthält

Analogien zwischen Schleifen: for-each/while

```
for ( Typ element : sammlung ) {
    anweisungen
}
```

≙

```
{
    final Iterator<Typ> i = sammlung.iterator();
    while (i.hasNext()) {
        Typ element = i.next();
        anweisungen
    }
}
```

```
for ( Typ element : array ) {
    anweisungen
}
```

≙

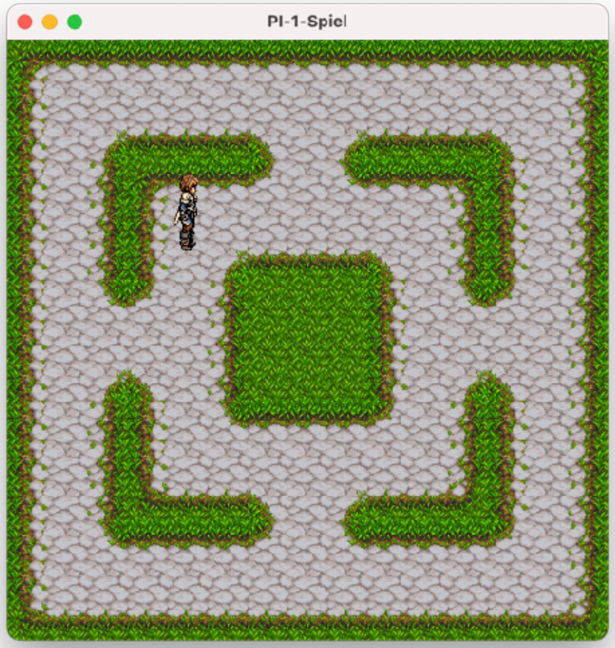
```
for (int i = 0; i < array.length; ++i) {
    Typ element = array[i];
    anweisungen
}
```

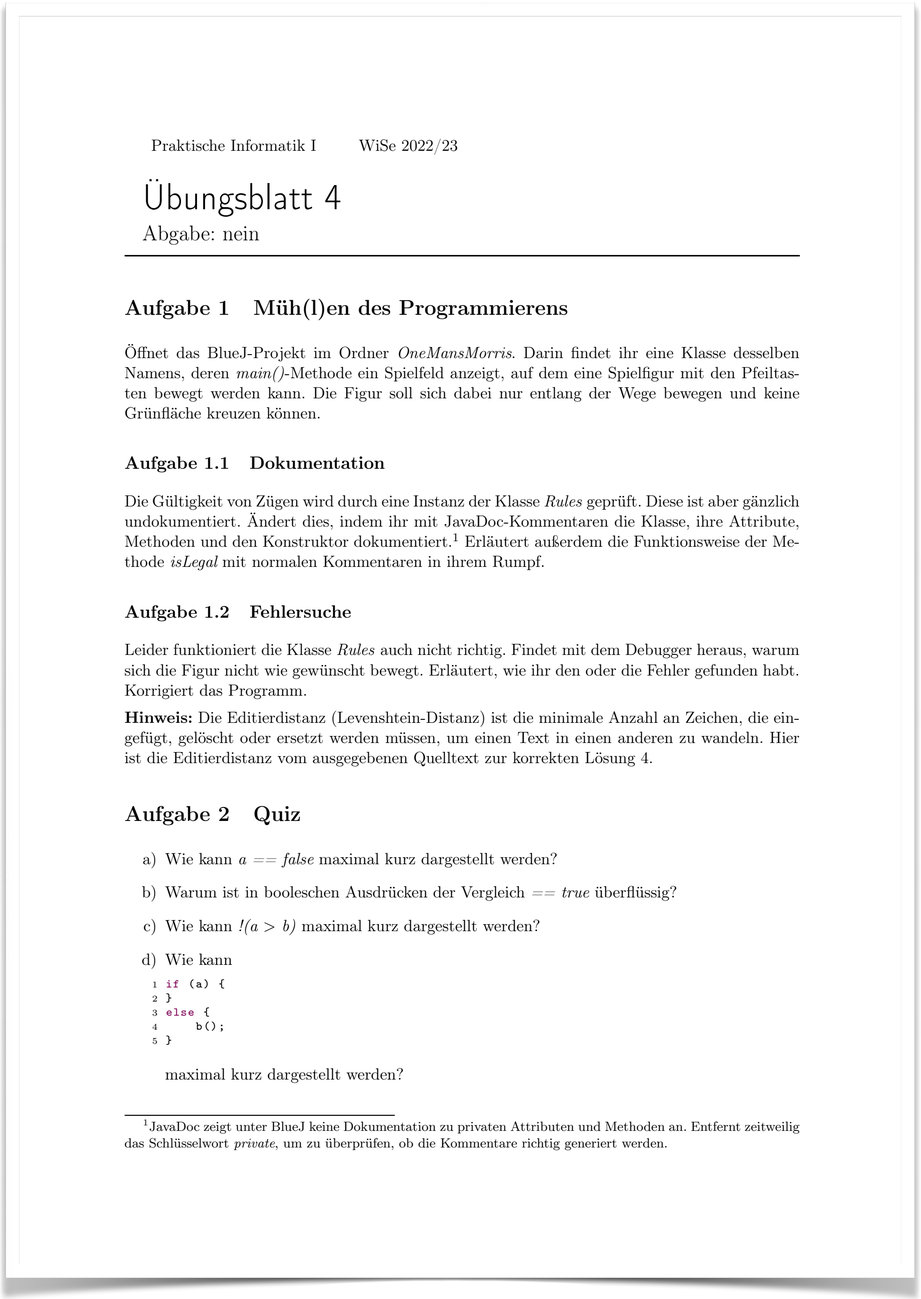
Zusammenfassung der Konzepte

- **Array**
- **Hinzurechnen**, **Pre**/**Post**-**Inkrement**/**Dekrement**
- **Bindungskraft** und **Assoziativität**
- **Zählschleife**

Übungsblatt 4

- Aufgabe 1
  - 1.1: Code verstehen und dokumentieren
  - 1.2: Debugger nutzen, um Fehler zu finden
- Aufgabe 2: Quiz





Praktische Informatik I WiSe 2022/23

Übungsblatt 4

Abgabe: nein

Aufgabe 1 Müh(l)en des Programmierens

Öffnet das BlueJ-Projekt im Ordner *OneMansMorris*. Darin findet ihr eine Klasse desselben Namens, deren *main()*-Methode ein Spielfeld anzeigt, auf dem eine Spielfigur mit den Pfeiltasten bewegt werden kann. Die Figur soll sich dabei nur entlang der Wege bewegen und keine Grünfläche kreuzen können.

Aufgabe 1.1 Dokumentation

Die Gültigkeit von Zügen wird durch eine Instanz der Klasse *Rules* geprüft. Diese ist aber gänzlich undokumentiert. Ändert dies, indem ihr mit JavaDoc-Kommentaren die Klasse, ihre Attribute, Methoden und den Konstruktor dokumentiert.[1] Erläutert außerdem die Funktionsweise der Methode *isLegal* mit normalen Kommentaren in ihrem Rumpf.

Aufgabe 1.2 Fehlersuche

Leider funktioniert die Klasse *Rules* auch nicht richtig. Findet mit dem Debugger heraus, warum sich die Figur nicht wie gewünscht bewegt. Erläutert, wie ihr den oder die Fehler gefunden habt. Korrigiert das Programm.

Hinweis: Die Editierdistanz (Levenshtein-Distanz) ist die minimale Anzahl an Zeichen, die eingefügt, gelöscht oder ersetzt werden müssen, um einen Text in einen anderen zu wandeln. Hier ist die Editierdistanz vom ausgegebenen Quelltext zur korrekten Lösung 4.

Aufgabe 2 Quiz

a) Wie kann $a == false$ maximal kurz dargestellt werden?

b) Warum ist in booleschen Ausdrücken der Vergleich $== true$ überflüssig?

c) Wie kann $!(a > b)$ maximal kurz dargestellt werden?

d) Wie kann

```
if (a) {
}
else {
    b();
}
```

maximal kurz dargestellt werden?

[1] JavaDoc zeigt unter BlueJ keine Dokumentation zu privaten Attributen und Methoden an. Entfernt zeitweilig das Schlüsselwort *private*, um zu überprüfen, ob die Kommentare richtig generiert werden.