Karlsruher Institut für Technologie

Programmieren – Wintersemester 2023

PD Dr. rer. nat. Robert Heinrich
Institute of Information Security and Dependability (KASTEL)
programmieren-vorlesung@cs.kit.edu

Beispielaufgaben Präsenzübung

Organisatorische Hinweise

Die Teilnahme setzt eine ordnungsgemäße Anmeldung für den Programmieren-Übungsschein für das laufende Semester voraus. Die Anmeldung für den Übungsschein ist nur bis zum *13.12.2023, um 12:00 Uhr* möglich.

Für die Präsenzübung sind *keine weiteren Hilfsmittel*, außer dokumentenechte Stifte in den Farben blau oder schwarz, gültige Ausweise und die zuvor geprüften Wörterbücher am Platz erlaubt. Zur Identitätsprüfung ist die gültige *Studierendenkarte* (Studienausweis) und ein gültiger *amtlicher Lichtbildausweis* (europäischer Personalausweis, elektronischer Aufenthaltstitel oder internationaler Reisepass) ebenso mitzubringen.

Sie müssen sich die Nummer des Ihnen zugeteilten Tutoriums merken, da Sie diese Nummer auf das Deckblatt der Präsenzübung schreiben müssen. Die Nummer des Ihnen zugeteilten Tutoriums können Sie unter anderem auch über das Wiwi-Portal[1] herausfinden.

Die Präsenzübung findet in der Regel gleichzeitig in mehreren Hörsälen auf dem *KIT-Campus Süd* und in diesen nacheinander in mehreren Sitzungen statt. Informationen zur persönlichen Hörsaaleinteilung und Sitzung werden in der Regel spätestens fünf Werktage nach dem Ende der Anmeldefrist des Übungsscheines über die *SDQ-NewsList*[2] bekannt gegeben. Falls Sie zu dem angebenden Zeitpunkt noch keine Informationen dort vorfinden können, jedoch sicher für den Übungsschein angemeldet sind, wenden Sie sich unverzüglich per E-Mail[3] an uns.

Bitte seien Sie bereits 10 Minuten vor Beginn Ihrer zugeteilten Sitzung vor Ihrem Hörsaal und warten Sie leise davor, bis Sie hereingebeten werden. Betreten Sie den Hörsaal erst, wenn die vorherige Sitzung beendet ist.

Die Aufgaben werden in der Präsenzübung nur auf Deutsch gestellt. Falls Sie ein Wörterbuch benötigen, geben Sie dieses bis spätestens Ende der Anmeldefrist des Übungsscheines im gemeinsamen Sekretariat von Prof. Koziolek und Prof. Reussner (Gebäude 50.34, Raum 328. Öffnungszeiten Montag bis Freitag, 11:00 Uhr bis 12:30 Uhr) zur Überprüfung ab. *Nicht durch uns geprüfte Wörterbücher können nicht verwendet werden.* Sie erhalten das Wörterbuch in der Präsenzübung von uns zurück. Falls Sie nicht teilnehmen, können Sie es wieder im Sekretariat abholen.

[1] https://portal.wiwi.kit.edu/ys
[2] https://s.kit.edu/newslist
[3] programmieren-vorlesung@cs.kit.edu

Dictionaries

All questions in the written test ("Präsenzübung") will be in German. If you need a dictionary, hand it in for checking at the joint secretary's office of Professor Koziolek and Professor Reussner (building 50.34, room 328. Opening hours: Monday to Friday, 11:00 to 12:30) by the end of the registration period for the training certificate ("Übungsschein") at the latest. *If we have not checked the dictionary, you will not be allowed to use it during the test.* We will return the dictionary to you immediately before the written test. If you don't participate you can pick the dictionary up at the secretary's office.

Aufbau der Präsenzübung

Die Präsenzübung prüft Wissen ab, dass Sie bei der eigenständigen Bearbeitung der Übungsblätter erworben haben. Die Präsenzübung ist auf den Umgang mit der Sprache Java und Implementierungswissen fokussiert. Um dieses Wissen nachzuweisen, müssen Sie einerseits kleine Quelltextausschnitte schreiben oder ergänzen, und andererseits das Verhalten von Quelltext oder eventuelle Fehler in kleinen Quelltextausschnitten analysieren.

Das bedeutet insbesondere, dass *kein* in der Vorlesung behandeltes Hintergrundwissen abgeprüft wird, wie beispielsweise Details zur IEEE-Gleitkommazahlendarstellung. Weiterhin müssen Sie keine Schnittstellen von Methoden der Java-API oder Signaturen von Methoden auswendig lernen. Falls nötig, werden diese in den Aufgaben angegeben. Die Präsenzübung wird unter anderem Aufgaben aus den folgenden Kategorien enthalten:

Java-Grundlagen: Grundlegende Java-Syntax wie Deklaration von Variablen, Klassen, Enum, Methoden, Sichtbarkeit, Überschattung und Gültigkeitsbereiche von Variablen, Zugriff auf Felder (Arrays) oder Attribute, Logische Operatoren. Einfache Arithmetik, Modulo, String-Konkatenation. ...

Kontrollfluss: `for`-, `while`- und `do-while`-Schleifen. `if`, `else-if`, `else`-Anweisungen. Kombination: in `for`-Schleife Eigenschaft von Element prüfen, verschachtelte `for`-Schleifen. Iterieren über Einträge in einem Feld. Einfache Rekursion. ...

Objekt-Orientierung: Konstruktoren. Sichtbarkeit. Zugriffsfunktionen. Vererbungsbeziehungen. Schnittstellen. Überschreiben und überladen von Methoden. Polymorphie. Statische und dynamische Bindung. ...

Die Beispielaufgaben in diesem Dokument sind jeweils einer der Kategorien zugeordnet. *Die Beispielaufgaben werden nicht abgegeben oder korrigiert.* Die Anzahl der Beispielaufgaben entspricht ausdrücklich nicht der Anzahl der Aufgaben, welche in der Präsenzübung bearbeitet werden müssen. In der Regel besteht die Präsenzübung aus *ca. 5 Aufgaben* und hat eine Bearbeitungszeit von *ca. 20 Minuten*. Genaue Informationen zum Umfang und der Bearbeitungszeit der Präsenzübung werden rechtzeitig im zugehörigen *ILIAS-Arbeitsbereich*[4] bekannt gegeben. Im Folgenden finden Sie neben weiteren allgemeinen Hinweisen ebenso Aufgaben, welche beispielhaft für die Aufgaben in der Präsenzübung sind.

[4] `https://s.kit.edu/ilias`

Bearbeitungshinweise

Für eine korrekte Lösung müssen nicht alle Lücken ausgefüllt werden. Die Größe der Lücken steht weiterhin nicht unbedingt in Relation zu den erreichbaren Punkten oder zur Länge des Textes, der für eine korrekte Lösung eingefüllt werden muss. Der von Ihnen eingefüllte Text muss kompilierbar sein. Verwenden Sie nur Elemente des Pakets `java.lang` der Version *Java SE 17*. Achten Sie im Folgenden auch auf Groß- und Kleinschreibung. Setzen Sie *nur* die in den folgenden Aufgabenstellungen angegebenen Informationen um. Sie müssen bei der Beantwortung der Präsenzübungsaufgaben *keine* Vorgaben zu Checkstyle-Regeln einhalten wie sonst bei der Bearbeitung der Übungsblätter.

Abschnitt I: Aufgaben

A.1 Java-Basics

(Lösung auf Seite 14)

Ergänzen Sie die folgende Klasse so, dass gilt: `theText` ist ein öffentliches Attribut (*field*), das eine Zeichenfolge enthält, `numbers` ist ein privates Attribut, das ein Feld (*array*) von Ganzzahlen enthält.

```
class ValueHolder {
    ____________ ____________ theText;
                 ____________ numbers;
}
```

A.2 Java-Basics

(Lösung auf Seite 14)

Ergänzen Sie die folgende Schnittstelle um eine Methode `doSomething`, die ein Feld (*array*) von Ganzzahlen entgegennimmt und keine Rückgabe hat.

```
interface Worker {
    ____________ ____________ ____________ doSomething(____________ ____________ ____________);
}
```

A.3 Kontrollfluss

(Lösung auf Seite 15)

Ergänzen Sie die folgenden Methode so, dass alle Werte des Felds `values` ausgegeben werden, die echt größer als `threshold` sind. Verwenden Sie zur Ausgabe die Methode `System.out.println`. Es gilt immer `values` $\neq$ `null`.

```
public static void printGreaterThan(int[] values, int threshold) {

}
```

A.4 Kontrollfluss

(Lösung auf Seite 15)

Ergänzen Sie den folgenden Code so, dass beim Aufruf von `doNTimes(int)` die Methode `doSomething()` n-mal aufgerufen wird. Es gilt immer `n` ≥ 0.

```
public static void doSomething() { /* ... */ }

public static void doNTimes(int n) {

}
```

A.5 Kontrollfluss

(Lösung auf Seite 16)

Ergänzen Sie den folgenden Code so, dass jedes zweite Element des Felds (*array*) `values` ausgegeben wird, beginnend mit dem zweiten Eintrag. Für das Feld `new int[] { 42, 5, 17, 7, 3, 1, 5, 10 }` wäre dies beispielsweise `5, 7, 1, 10`. Es gilt immer `values` ≠ `null`.

```
public static void printSkip(int[] values) {
    for (int i =              ;                       ;              ) {

        System.out.println(values[i]);

    }
}
```

A.6 Kontrollfluss

(Lösung auf Seite 16)

Ergänzen Sie den folgenden Code so, dass nacheinander die Werte in `values` ausgegeben werden, die an den in `positions` definierten Stellen stehen. Sie können davon ausgehen, dass `positions` nur Werte zwischen einschließlich `0` und `values.length - 1` enthält. Es gilt immer `values` ≠ `null` und `positions` ≠ `null`.

```
public static void printValuesAt(String[] values, int[] positions)
   {
    for (int i =              ;                       ;              ) {
        System.out.println(                                          );
    }
}
```

A.7 Kontrollfluss

(Lösung auf Seite 17)

Ergänzen Sie den folgenden Code so, dass alle ungeraden Werte in `values` ausgegeben werden (mit `System.out.println`). Sie können davon ausgehen, dass `values` nicht `null` ist.

```
public static void printOdds(int values[]) {
    for (int i =              ;                       ;              ) {
        
        
        
        
        
        
        
        
    }
}
```

A.8 Kontrollfluss

(Lösung auf Seite 17)

Es sei die Methode `evaluate()` mit dem Rückgabetyp `boolean` gegeben. Ergänzen Sie den folgenden Code so, dass die Methode `evaluate()` so lange aufgerufen wird, bis sie zum ersten Mal `true` zurückgibt. Zählen Sie in der Variable `i`, wie oft in Folge `false` zurückgegeben wurde.

```
int i = 0;
while (                                    ) {
    
    
    
    
}
System.out.println(i);
```

A.9 Kontrollfluss

(Lösung auf Seite 18)

Ergänzen Sie den folgenden Quelltext der Methode `doSomething(int)` so, dass diese wenn `i` ungerade ist, `i` um 1 erhöht und wenn `i` gerade und größer als oder gleich 3 ist, den Text „Fall 2“ aus gibt.

```
public static int doSomething(int i) {
    if (                                        ) {
        
        System.out.println("Fall 2");
    }

    return i;
}
```

A.10 Objekt-Orientierung – Attribute

(Lösung auf Seite 18)

Ergänzen Sie die Klasse `SuperCollider` mit einem öffentlichen statischen Attribut (*field*) `myValue` das eine Ganzzahl (*integer*) enthalten kann und einem privaten Attribut `myText`, das einen String enthalten kann.

```
class SuperCollider {
    
}
```

A.11 Objekt-Orientierung – Objekterzeugung

(Lösung auf Seite 19)

Es sei eine Klasse `Date` mit den beiden Konstruktoren `Date(int dayOfYear)` und `Date(int day, int month)` gegeben. Ergänzen Sie den folgenden Code so, dass `date0` mit dem 42-sten Tag des Jahres (also `dayOfYear = 42`) belegt wird und `date1` mit dem Datum mit `day = 7` und `month = 10`.

```
Date date0 =                                    ;
Date date1 =                                    ;
```

A.12 Objekt-Orientierung – Getter und Setter

(Lösung auf Seite 19)

Ergänzen Sie die folgende Klassendefinition um den Setter `setValueAt` und `getValueAt`, die jeweils auf das Attribut (*field*) `values` und den Wert an der Stelle `pos` zugreifen. Sie können davon ausgehen, dass `values` immer ungleich `null` ist und für das Argument gilt: `0 <= pos < values.length`.

```
class MagicSuperCompiler {
    private String[] values;

    public void setValueAt(int pos, String newValue) {
        
    }

    public String getValueAt(int pos) {
        
    }
}
```

A.13 Objekt-Orientierung – Getter und Setter

(Lösung auf Seite 20)

Ergänzen Sie die folgende Klassendefinition um einen Getter und einen Setter für das Attribut `value`. Setzen Sie im Setter nur neue Werte, die größer gleich dem Attribut (*field*) `min` und kleiner gleich dem Attribut `max` sind.

```
class MagicSuperCompiler {
    private int min;
    private int max;

    private int value;

    // Getter for value


    // Setter for value


}
```

A.14 Objekt-Orientierung – Vererbung

(Lösung auf Seite 21)

Geben Sie für jede markierte Zeile den Rückgabewert des Ausdrucks an. Falls die Zeile zu einem Kompilierfehler führt, füllen Sie die Lücke mit „Fehler“.

```
class Parent {
    public String getFoo() { return "parentFoo"; }
}
class Child extends Parent {
    public String getFoo() { return "childFoo"; }
    public String getBar() { return "childBar"; }
}
class GrandChild extends Child {
    public String getFoo() { return "grandChildFoo"; }
}

Parent p0 = new Child();
p0.getFoo();
p0.getBar();
Child c0 = new GrandChild();
c0.getFoo();
c0.getBar();
GrandChild gc0 = new GrandChild();
gc0.getFoo();
gc0.getBar();
```

A.15 Objekt-Orientierung – Vererbung

(Lösung auf Seite 22)

Markieren Sie im folgenden Code die Zeilen, in denen unzulässige Aufrufe oder Zuweisungen stehen, d.h. Aufrufe oder Zuweisungen, die zu Kompilierfehlern führen.

```
interface IfA { void doA(); }
interface IfB { void doB(); }

class Parent implements IfA {
    public void doA() {}
    public void doC() {}
}

Parent p0 = new Parent();
IfA ifa0 = new Parent();
p0.doA();
p0.doB();
p0.doC();
ifa0.doA();
ifa0.doB();
ifa0.doC();
```

A.16 Objekt-Orientierung – Schnittstellen und Schnittstellenimplementierung

(Lösung auf Seite 23)

Schreiben Sie Java-Code für eine Schnittstelle, die eine Methodensignatur Ihrer Wahl enthält und eine Klasse, die diese Schnittstelle implementiert. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

A.17 Objekt-Orientierung – Konstruktor

(Lösung auf Seite 24)

Ergänzen Sie die folgende Klassendefinition mit einem Konstruktor, der ein Feld (*array*) `String[]` entgegennimmt und die Werte mit dem Operator `+` verbindet (konkateniert) und in das Attribut (*field*) `value` schreibt.

```
class MagicSuperCompiler {
    private String value;

    public MagicSuperCompiler(String value) {
        this.value = value;
    }

    public MagicSuperCompiler(String[] values) {
        
        for (int i = 0; i < values.length; i++) {
            
        }
        
    }
}
```

Abschnitt II: Lösungen

L.1 Java-Basics

(Aufgabe auf Seite 3)

Ergänzen Sie die folgende Klasse so, dass gilt: `theText` ist ein öffentliches Attribut (*field*), das eine Zeichenfolge enthält, `numbers` ist ein privates Attribut, das ein Feld (*array*) von Ganzzahlen enthält.

```
class ValueHolder {
    public String theText; // oder CharSequence
    private int[] numbers; // oder long[], Integer[], ...
}
```

L.2 Java-Basics

(Aufgabe auf Seite 3)

Ergänzen Sie die folgende Schnittstelle um eine Methode `doSomething`, die ein Feld (*array*) von Ganzzahlen entgegennimmt und keine Rückgabe hat.

```
interface Worker {
    void doSomething(int[] values);
    // kein Fehler, falls public, oder long[], Integer[], ...
}
```

L.3 Kontrollfluss

(Aufgabe auf Seite 4)

Ergänzen Sie die folgenden Methode so, dass alle Werte des Felds `values` ausgegeben werden, die echt größer als `threshold` sind. Verwenden Sie zur Ausgabe die Methode `System.out.println`. Es gilt immer `values` $\neq$ `null`.

```
public static void printGreaterThan(int[] values, int threshold) {
    // Antwortmöglichkeit A
    for (int i = 0; i < values.length; i++) {
        if (values[i] > threshold) {
            System.out.println(values[i]);
        }
    }

    // Antwortmöglichkeit B
    for (int value : values) {
        if (value > threshold) {
            System.out.println(value);
        }
    }
}
```

L.4 Kontrollfluss

(Aufgabe auf Seite 4)

Ergänzen Sie den folgenden Code so, dass beim Aufruf von `doNTimes(int)` die Methode `doSomething()` n-mal aufgerufen wird. Es gilt immer `n` ≥ 0.

```
public static void doSomething() { /* ... */ }

public static void doNTimes(int n) {
    for (int i = 0; i < n; i++) {
        doSomething();
    }
}
```

L.5 Kontrollfluss

(Aufgabe auf Seite 5)

Ergänzen Sie den folgenden Code so, dass jedes zweite Element des Felds (*array*) `values` ausgegeben wird, beginnend mit dem zweiten Eintrag. Für das Feld `new int[] { 42, 5, 17, 7, 3, 1, 5, 10 }` wäre dies beispielsweise `5, 7, 1, 10`. Es gilt immer `values` ≠ `null`.

```
public static void printSkip(int[] values) {
    for (int i = 1; i < values.length; i += 2) {
        System.out.println(values[i]);
    }

    // oder

    for (int i = 0; i < values.length; i++) {
        if ((i % 2) == 1) {
            System.out.println(values[i]);
        }
    }
}
```

L.6 Kontrollfluss

(Aufgabe auf Seite 5)

Ergänzen Sie den folgenden Code so, dass nacheinander die Werte in `values` ausgegeben werden, die an den in `positions` definierten Stellen stehen. Sie können davon ausgehen, dass `positions` nur Werte zwischen einschließlich `0` und `values.length - 1` enthält. Es gilt immer `values` ≠ `null` und `positions` ≠ `null`.

```
public static void printValuesAt(String[] values, int[] positions)
   {
    for (int i = 0; i < positions.length; i++) {
        System.out.println(values[positions[i]]);
    }
}
```

L.7 Kontrollfluss

(Aufgabe auf Seite 6)

Ergänzen Sie den folgenden Code so, dass alle ungeraden Werte in `values` ausgegeben werden (mit `System.out.println`). Sie können davon ausgehen, dass `values` nicht `null` ist.

```
public static void printOdds(int values[]) {
    for (int i = 0; i < values.length; i++) {
        if ((values[i] % 2) == 1) {
            System.out.println(values[i]);
        }
    }
}
```

L.8 Kontrollfluss

(Aufgabe auf Seite 6)

Es sei die Methode `evaluate()` mit dem Rückgabetyp `boolean` gegeben. Ergänzen Sie den folgenden Code so, dass die Methode `evaluate()` so lange aufgerufen wird, bis sie zum ersten Mal `true` zurückgibt. Zählen Sie in der Variable `i`, wie oft in Folge `false` zurückgegeben wurde.

```
int i = 0;
while (!evaluate()) {
    i++;
}
System.out.println(i);
```

L.9 Kontrollfluss

(Aufgabe auf Seite 7)

Ergänzen Sie den folgenden Quelltext der Methode `doSomething(int)` so, dass diese wenn `i` ungerade ist, `i` um 1 erhöht und wenn `i` gerade und größer als oder gleich 3 ist, den Text „Fall 2“ aus gibt.

```
public static int doSomething(int i) {
    if ((i % 2) == 1) {
        i++;
    } else if (i >= 3) { // Nur if ist hier nicht korrekt
        System.out.println("Fall 2");
    }

    return i;
}
```

L.10 Objekt-Orientierung – Attribute

(Aufgabe auf Seite 7)

Ergänzen Sie die Klasse `SuperCollider` mit einem öffentlichen statischen Attribut (*field*) `myValue` das eine Ganzzahl (*integer*) enthalten kann und einem privaten Attribut `myText`, das einen String enthalten kann.

```
class SuperCollider {
    public static int myValue;
    private String myText;
}
```

L.11 Objekt-Orientierung – Objekterzeugung

(Aufgabe auf Seite 8)

Es sei eine Klasse `Date` mit den beiden Konstruktoren `Date(int dayOfYear)` und `Date(int day, int month)` gegeben. Ergänzen Sie den folgenden Code so, dass `date0` mit dem 42-sten Tag des Jahres (also `dayOfYear = 42`) belegt wird und `date1` mit dem Datum mit `day = 7` und `month = 10`.

```
Date date0 = new Date(42);
Date date1 = new Date(7, 10);
```

L.12 Objekt-Orientierung – Getter und Setter

(Aufgabe auf Seite 8)

Ergänzen Sie die folgende Klassendefinition um den Setter `setValueAt` und `getValueAt`, die jeweils auf das Attribut (*field*) `values` und den Wert an der Stelle `pos` zugreifen. Sie können davon ausgehen, dass `values` immer ungleich `null` ist und für das Argument gilt: `0 <= pos < values.length`.

```
class MagicSuperCompiler {
    private String[] values;

    public void setValueAt(int pos, String newValue) {
        this.values[pos] = newValue;
    }

    public String getValueAt(int pos) {
        return this.values[pos];
    }
}
```

L.13 Objekt-Orientierung – Getter und Setter

(Aufgabe auf Seite 9)

Ergänzen Sie die folgende Klassendefinition um einen Getter und einen Setter für das Attribut `value`. Setzen Sie im Setter nur neue Werte, die größer gleich dem Attribut (*field*) `min` und kleiner gleich dem Attribut `max` sind.

```
class MagicSuperCompiler {
    private int min;
    private int max;

    private int value;

    // Getter for value
    public int getValue() {
        return this.value; // oder: return value;
    }

    // Setter for value
    public void setValue(int newValue) {
        if ((min <= newValue) && (newValue <= max)) {
            // oder alternative korrekte Bedingungen
            this.value = newValue; // oder: value = newValue;
        }
    }
}
```

L.14 Objekt-Orientierung – Vererbung

(Aufgabe auf Seite 10)

Geben Sie für jede markierte Zeile den Rückgabewert des Ausdrucks an. Falls die Zeile zu einem Kompilierfehler führt, füllen Sie die Lücke mit „Fehler“.

```
class Parent {
    public String getFoo() { return "parentFoo"; }
}
class Child extends Parent {
    public String getFoo() { return "childFoo"; }
    public String getBar() { return "childBar"; }
}
class GrandChild extends Child {
    public String getFoo() { return "grandChildFoo"; }
}

Parent p0 = new Child();
p0.getFoo(); childFoo
p0.getBar(); Fehler
Child c0 = new GrandChild();
c0.getFoo(); grandChildFoo
c0.getBar(); childBar
GrandChild gc0 = new GrandChild();
gc0.getFoo(); grandChildFoo
gc0.getBar(); childBar
```

L.15 Objekt-Orientierung – Vererbung

(Aufgabe auf Seite 11)

Markieren Sie im folgenden Code die Zeilen, in denen unzulässige Aufrufe oder Zuweisungen stehen, d.h. Aufrufe oder Zuweisungen, die zu Kompilierfehlern führen.

```
interface IfA { void doA(); }
interface IfB { void doB(); }

class Parent implements IfA {
    public void doA() {}
    public void doC() {}
}

Parent p0 = new Parent();
IfA ifa0 = new Parent();
  p0.doA();
✗ p0.doB();
  p0.doC();
  ifa0.doA();
✗ ifa0.doB();
✗ ifa0.doC();
```

L.16 Objekt-Orientierung – Schnittstellen und Schnittstellenimplementierung

(Aufgabe auf Seite 12)

Schreiben Sie Java-Code für eine Schnittstelle, die eine Methodensignatur Ihrer Wahl enthält und eine Klasse, die diese Schnittstelle implementiert. Sie müssen keine Funktionalität oder ähnliches beschreiben (d.h. Sie können Methodenrümpfe leer lassen).

```
public interface Worker {
    void work();
}

public class HardWorker implements Worker {
    public void work() {
    }
}
```

L.17 Objekt-Orientierung – Konstruktor

(Aufgabe auf Seite 13)

Ergänzen Sie die folgende Klassendefinition mit einem Konstruktor, der ein Feld (*array*) `String[]` entgegennimmt und die Werte mit dem Operator `+` verbindet (konkateniert) und in das Attribut (*field*) `value` schreibt.

```
class MagicSuperCompiler {
    private String value;

    public MagicSuperCompiler(String value) {
        this.value = value;
    }

    public MagicSuperCompiler(String[] values) {
        this.value = "";
        // oder value = "" oder andere Variable String val2
        for (int i = 0; i < values.length; i++) {
            this.value += values[i];
            // oder value += values[i] oder val2 += values[i]
        }
        // falls andere Variable dann this.value = val2;
    }
}
```