Willkommen

Willkommen

Letztes Mal

- Variablen
- Einfache Operationen
- Methoden - Crashkurs

Willkommen

Letztes Mal

- Variablen
- Einfache Operationen
- Methoden - Crashkurs

Dieses Mal

- Konstruktoren und `final`
- Programmablauf
- Methoden – Vertiefung
- Geheimnisprinzip – `public`, `protected`, `private`

Willkommen

Letztes Mal

- Variablen
- Einfache Operationen
- Methoden - Crashkurs

Dieses Mal

- Konstruktoren und `final`
- Programmablauf
- Methoden – Vertiefung
- Geheimnisprinzip – `public`, `protected`, `private`

Nächstes Mal

- Kontrollfluss
- Tests

Fazit: 1. Übungsblatt

Anmerkungen

- Es ist immer besser, wenn eure Lösung kompiliert
- Bezeichner sollen sorgfältig gewählt werden

Fazit: 1. Übungsblatt

Anmerkungen

- Es ist immer besser, wenn eure Lösung kompiliert
- Bezeichner sollen sorgfältig gewählt werden

Musterlösung

inklusive JavaDoc

Konstruktoren – Einleitung

Wir finden Initialisierung toll...

Konstruktoren – Einleitung

Wir finden Initialisierung toll...

da Attribute und Variablen dann direkt den gewollten Wert haben

Konstruktoren – Einleitung

Wir finden Initialisierung toll...

da Attribute und Variablen dann direkt den gewollten Wert haben
Um dies auch für komplexe Objekte ermöglichen zu können haben wir Konstruktoren!

Beispiel: Initialisierung von komplexen Zahlen

Konstruktoren – Einleitung

Wir finden Initialisierung toll...

da Attribute und Variablen dann direkt den gewollten Wert haben
Um dies auch für komplexe Objekte ermöglichen zu können haben wir Konstruktoren!

Beispiel: Initialisierung von komplexen Zahlen

Konstruktoren – Überblick

- dienen dazu ein Objekt zu initialisieren
- sind generell sehr ähnlich zu Methoden
- besitzen jedoch keinen Rückgabetyp und geben auch nichts zurück
- können aber Parameter haben

Konstruktoren – Beispiel

```
1  class Counter {
2      int count;
3
4      Counter() {
5          count = 1;
6      }
7
8      Counter(int startcount) {
9          count = startcount;
10     }
11 }
```

Konstruktoren – Wie funktioniert das?

`new`

`new` legt ein Objekt an und initialisert es mit dem passenden Konstruktor
Dieser wird ausgewählt aufgrund

- der Klasse
- der Parameteranzahl
- der Reinfolge der Parametertypen

Parameternamen sind nicht relevant!

Konstruktoren – Wie funktioniert das?

`new`

`new` legt ein Objekt an und initialisert es mit dem passenden Konstruktor
Dieser wird ausgewählt aufgrund

- der Klasse
- der Parameteranzahl
- der Reinfolge der Parametertypen

Parameternamen sind nicht relevant!

Beispiele

```
Counter countA = new Counter();

Counter countB = new Counter(0);
```

final

Einleitung

`final` dient zur Deklaration von Konstanten, die zur Laufzeit nicht verändert werden können
Es kann bei der Deklaration Attributen/Variablen vorangestellt werden
Wenn es in Verbindung mit `static` auftaucht, verwenden wir nur Großbuchstaben für den Bezeichner
Beispiele:

```
final static double PI = 3.14;

final double epsilon = 1E-20;
```

final

Einleitung

final dient zur Deklaration von Konstanten, die zur Laufzeit nicht verändert werden können
Es kann bei der Deklaration Attributen/Variablen vorangestellt werden
Wenn es in Verbindung mit static auftaucht, verwenden wir nur Großbuchstaben für den Bezeichner
Beispiele:

```
final static double PI = 3.14;

final double epsilon = 1E-20;
```

final und Konstruktoren

In Konstruktoren können (und müssen) konstante Attribute der Klasse gesetzt werden, sofern nicht bereits eine Zuweisung besteht

Konstruktoren – Beispiel mit `final`

```
1  class Complex {
2      final double re;
3      final double im;
4
5      Complex() {
6          re = 0;
7          im = 0;
8      }
9
10     Complex(double re) {
11         this.re = re;
12         im = 0;
13     }
14
15     Complex(double re, double im) {
16         this.re = re;
17         this.im = im;
18     }
19 }
```

Auto: Aufgabe 1

Programmablauf

```
1  public class Statistik {
2      double  summeX = 0;
3      double  summeX2 = 0;
4      int n = 0;
5
6      public void hinzufuegen(double x) {
7          summeX = summeX + x;
8          summeX2 = summeX2 + x * x;
9          n = n + 1;
10     }
11
12     public double leseMittelwert() {
13         return summeX / n;
14     }
15
16     public double leseStandardabweichung() {
17         double mittelwert = leseMittelwert();
18         double varianz = summeX2 / n - mittelwert * mittelwert;
19         return Math.sqrt(varianz);
20     }
21 }
```

Programmablauf

```
1  public class Statistik {
2      double  summeX = 0;
3      double  summeX2 = 0;
4      int n = 0;
5
6      public void hinzufuegen(double x) {
7          summeX = summeX + x;
8          summeX2 = summeX2 + x * x;
9          n = n + 1;
10     }
11
12     public double leseMittelwert() {
13         return summeX / n;
14     }
15
16     public double leseStandardabweichung() {
17         double mittelwert = leseMittelwert();
18         double varianz = summeX2 / n - mittelwert * mittelwert;
19         return Math.sqrt(varianz);
20     }
21 }
```

java Beispiel2Statistik

Was passiert?

Methoden – Vertiefung

Überladen

Wie bei Konstruktoren ist es auch bei Methoden möglich, Methoden mit gleichen Namen aufgrund von Parameteranzahl und Parametertypen zu unterscheiden

Beispiel

```
1  boolean isZero(int value) {
2      return value == 0;
3  }
4
5  boolean isZero(double value) {
6      return Math.abs(value) <= 1E-30;
7  }
8
9  boolean isZero(String value) {
10     return isZero(value.length());
11 }
```

Geheimnisprinzip

Ziel

Wiederverwendbarkeit und Wartbarkeit

Geheimnisprinzip

Ziel

Wiederverwendbarkeit und Wartbarkeit

Lokalitätsprinzip

Änderungen sollen nur lokale Auswirkungen haben

Geheimnisprinzip

Ziel

Wiederverwendbarkeit und Wartbarkeit

Lokalitätsprinzip

Änderungen sollen nur lokale Auswirkungen haben

Folgerung

Daher erlauben wir nur Zugriff (von außerhalb der Klasse) auf Attribute, Konstanten und Methoden wenn es erforderlich ist

`public`	Zugriff aus jeder Klasse möglich
`protected`	Zugriff von innerhalb der Klasse und aus Unterklassen (später) erlaubt
`private`	Zugriff nur von innerhalb der Klasse erlaubt

Geheimnisprinzip – Beispiel

```
1  public class Counter {
2      private int count;
3
4      public Counter() {
5          count = 1;
6      }
7
8      public Counter(int count) {
9          this.count = count;
10     }
11
12     public int getCount() {
13         return count;
14     }
15
16     public void incrementCounter() {
17         count = count + 1;
18     }
19
20     public void decrementCounter() {
21         count = count  - 1;
22     }
23
24     public boolean isZero() {
25         return count == 0;
26     }
27 }
```

Auto: Aufgabe 2

Ende

TODO

1. Einreichen einer Lösung für das 2. Übungsblatt im Praktomat bis 22.11.2010, 13:00
2. Anmelden für den Übungsschein auf https://studium.kit.edu/ bis 31.3.2011

Vielen Dank für die Aufmerksamkeit!

...und viel Spaß beim Programmieren :)