

Globalübung 22.11.2005

- 1 Methoden
- 2 Das Schlüsselwort `static`
- 3 Schnittstellendokumentation
- 4 Javadoc

Überblick

- 1 Methoden
- 2 Das Schlüsselwort `static`
- 3 Schnittstellendokumentation
- 4 Javadoc

Methoden

Vorteile:

- modularer Code
- wartbarer Code
- wiederverwendbarer Code
- lesbarer Code

Methoden

Vorteile:

- modularer Code
(klar abgegrenzte Teilalgorithmen)
- wartbarer Code
- wiederverwendbarer Code
- lesbarer Code

Methoden

Vorteile:

- modularer Code
(klar abgegrenzte Teilalgorithmen)
- wartbarer Code
(Fehler nur einmal in Methode korrigieren)
- wiederverwendbarer Code
- lesbarer Code

Methoden

Vorteile:

- modularer Code
(klar abgegrenzte Teilalgorithmen)
- wartbarer Code
(Fehler nur einmal in Methode korrigieren)
- wiederverwendbarer Code
(Methodenaufruf statt Copy & Paste)
- lesbarer Code

Methoden

Vorteile:

- modularer Code
(klar abgegrenzte Teilalgorithmen)
- wartbarer Code
(Fehler nur einmal in Methode korrigieren)
- wiederverwendbarer Code
(Methodenaufruf statt Copy & Paste)
- lesbarer Code
(Methodenname statt Algorithmus)

Methoden

Vorteile:

- modularer Code
(klar abgegrenzte Teilalgorithmen)
- wartbarer Code
(Fehler nur einmal in Methode korrigieren)
- wiederverwendbarer Code
(Methodenaufruf statt Copy & Paste)
- lesbarer Code
(Methodenname statt Algorithmus)

Nachteile:

- Gefahr: unerwünschte / undokumentierte Seiteneffekte

Überblick

- 1 Methoden
- 2 Das Schlüsselwort `static`**
- 3 Schnittstellendokumentation
- 4 Javadoc

Nicht-statische Attribute

```
public class A {  
    ...  
    int someInt;  
    A someObject;  
    ...  
}
```

- jedes Objekt der Klasse hat seine eigenen Attribute
 - ⇒ Zugriff auf Attribut nur über Objekte möglich
 - ⇒ Änderungen betreffen nur Attribute des Objekts und nicht Attribute anderer Objekte der Klasse (beachte jedoch Referenzen)
- Einsatz:
lokale Eigenschaften, ...

Nicht-statische Attribute

```
public class A {  
    ...  
    int someInt;  
    A someObject;  
    ...  
}
```

- jedes Objekt der Klasse hat seine eigenen Attribute
 - ⇒ Zugriff auf Attribut nur über Objekte möglich
 - ⇒ Änderungen betreffen nur Attribute des Objekts und nicht Attribute anderer Objekte der Klasse (beachte jedoch Referenzen)
- Einsatz:
lokale Eigenschaften, ...

Statische Attribute

```
public class A {  
    ...  
    static int someInt;  
    static A someObject;  
    ...  
}
```

- sind einmalig für die gesamte Klasse
- ⇒ Zugriff auf Attribut über Klasse möglich
man benötigt keine Objekte zum Zugriff
- ⇒ Änderungen betreffen alle Objekte der Klasse
- Einsatz:
globale Zähler, Konstanten, ...

Statische Attribute

```
public class A {  
    ...  
    static int someInt;  
    static A someObject;  
    ...  
}
```

- sind einmalig für die gesamte Klasse
- ⇒ Zugriff auf Attribut über Klasse möglich
man benötigt keine Objekte zum Zugriff
- ⇒ Änderungen betreffen alle Objekte der Klasse
- Einsatz:
globale Zähler, Konstanten, ...

Nicht-statische Methoden (`someMethod`)

```
public class A {  
    int nr;  
    static int staticNr;  
    ...  
    A someMethod() { ... }  
    ...  
    static void otherMethod(A someA) { ... }  
    ...  
}
```

- Aufruf nur über Objekte möglich
- direkter Zugriff auf alle Attribute der Klasse
- direkter Zugriff auf alle Methoden der Klasse

Statische Methoden (`someMethod`)

```
public class A {  
    int nr;  
    static int staticNr;  
    ...  
    static A someMethod() { ... }  
    ...  
    void otherMethod(A someA) { ... }  
    ...  
}
```

- Aufruf über Klassennamen möglich
- ⇒ benötigt kein Objekt der Klasse zum Aufruf
- direkter Zugriff nur auf statische Attribute der Klasse
 - direkter Zugriff nur auf statische Methoden der Klasse

Überblick

- 1 Methoden
- 2 Das Schlüsselwort `static`
- 3 Schnittstellendokumentation
- 4 Javadoc

Schnittstellendokumentation (S.)

S. für z.B. Klasse `Zeit` gegeben (+ genaue Beschreibung):

```
class Zeit:  
public Zeit(int stunden, int minuten)  
public int getStunden  
public int getMinuten  
public boolean frueher(Zeit z)  
public void addiereMinuten(int minuten)
```

- S. stellen genaue Anforderung an zu implementierende Klasse
- Erlaubt unabhängige Entwicklung von nutzender und implementierender Klasse
- Implementierungen können unabhängig getestet werden
- Implementierungen können ausgetauscht werden

⇒ Modularität

⇒ Wiederverwendbarkeit

⇒ parallele Entwicklung möglich

Schnittstellendokumentation (S.)

S. für z.B. Klasse `Zeit` gegeben (+ genaue Beschreibung):

```
class Zeit:  
public Zeit(int stunden, int minuten)  
public int getStunden  
public int getMinuten  
public boolean frueher(Zeit z)  
public void addiereMinuten(int minuten)
```

- S. stellen genaue Anforderung an zu implementierende Klasse
- Erlaubt unabhängige Entwicklung von nutzender und implementierender Klasse
- Implementierungen können unabhängig getestet werden
- Implementierungen können ausgetauscht werden

⇒ Modularität

⇒ Wiederverwendbarkeit

⇒ parallele Entwicklung möglich

Schnittstellendokumentation (S.)

S. für z.B. Klasse `Zeit` gegeben (+ genaue Beschreibung):

```
class Zeit:  
public Zeit(int stunden, int minuten)  
public int getStunden  
public int getMinuten  
public boolean frueher(Zeit z)  
public void addiereMinuten(int minuten)
```

- S. stellen genaue Anforderung an zu implementierende Klasse
- Erlaubt unabhängige Entwicklung von nutzender und implementierender Klasse
- Implementierungen können unabhängig getestet werden
- Implementierungen können ausgetauscht werden

⇒ Modularität

⇒ Wiederverwendbarkeit

⇒ parallele Entwicklung möglich

Schnittstellendokumentation (S.)

S. für z.B. Klasse `Zeit` gegeben (+ genaue Beschreibung):

```
class Zeit:  
public Zeit(int stunden, int minuten)  
public int getStunden  
public int getMinuten  
public boolean frueher(Zeit z)  
public void addiereMinuten(int minuten)
```

- S. stellen genaue Anforderung an zu implementierende Klasse
- Erlaubt unabhängige Entwicklung von nutzender und implementierender Klasse
- Implementierungen können unabhängig getestet werden
- Implementierungen können ausgetauscht werden

⇒ Modularität

⇒ Wiederverwendbarkeit

⇒ parallele Entwicklung möglich

Schnittstellendokumentation (S.)

S. für z.B. Klasse `Zeit` gegeben (+ genaue Beschreibung):

```
class Zeit:  
public Zeit(int stunden, int minuten)  
public int getStunden  
public int getMinuten  
public boolean frueher(Zeit z)  
public void addiereMinuten(int minuten)
```

- S. stellen genaue Anforderung an zu implementierende Klasse
- Erlaubt unabhängige Entwicklung von nutzender und implementierender Klasse
- Implementierungen können unabhängig getestet werden
- Implementierungen können ausgetauscht werden

⇒ Modularität

⇒ Wiederverwendbarkeit

⇒ parallele Entwicklung möglich

Schnittstellendokumentation (S.)

S. für z.B. Klasse `Zeit` gegeben (+ genaue Beschreibung):

```
class Zeit:  
public Zeit(int stunden, int minuten)  
public int getStunden  
public int getMinuten  
public boolean frueher(Zeit z)  
public void addiereMinuten(int minuten)
```

- S. stellen genaue Anforderung an zu implementierende Klasse
 - Erlaubt unabhängige Entwicklung von nutzender und implementierender Klasse
 - Implementierungen können unabhängig getestet werden
 - Implementierungen können ausgetauscht werden
- ⇒ Modularität
- ⇒ Wiederverwendbarkeit
- ⇒ parallele Entwicklung möglich

Überblick

- 1 Methoden
- 2 Das Schlüsselwort `static`
- 3 Schnittstellendokumentation
- 4 Javadoc

Javadoc

- Schnittstellendokumentationen müssen klar definieren, was die jeweiligen Methoden leisten sollen.

⇒ Beschreibung für jede Methode ist notwendig

⇒ Beschreibung kann auch direkt als Java-Kommentar vor jeder Methode geschrieben werden. Nutze speziellen Kommentar:

```
/**  
 * Beschreibung  
 * ...  
 */
```

- *Javadoc* kann nun HTML-Seiten daraus generieren.

⇒ Dokumentation für Nutzer und Entwickler der Klassen.

- Spezialbefehle in der Beschreibung für Referenzen, Parameter-Beschreibung, ...

Javadoc

- Schnittstellendokumentationen müssen klar definieren, was die jeweiligen Methoden leisten sollen.
- ⇒ Beschreibung für jede Methode ist notwendig
- ⇒ Beschreibung kann auch direkt als Java-Kommentar vor jeder Methode geschrieben werden. Nutze speziellen Kommentar:

```
/**  
 * Beschreibung  
 * ...  
 */
```
- *Javadoc* kann nun HTML-Seiten daraus generieren.
- ⇒ Dokumentation für Nutzer und Entwickler der Klassen.
- Spezialbefehle in der Beschreibung für Referenzen, Parameter-Beschreibung, ...

Javadoc

- Schnittstellendokumentationen müssen klar definieren, was die jeweiligen Methoden leisten sollen.
- ⇒ Beschreibung für jede Methode ist notwendig
- ⇒ Beschreibung kann auch direkt als Java-Kommentar vor jeder Methode geschrieben werden. Nutze speziellen Kommentar:

```
/**  
 * Beschreibung  
 * ...  
 */
```

- *Javadoc* kann nun HTML-Seiten daraus generieren.
- ⇒ Dokumentation für Nutzer und Entwickler der Klassen.
- Spezialbefehle in der Beschreibung für Referenzen, Parameter-Beschreibung, ...

Javadoc

- Schnittstellendokumentationen müssen klar definieren, was die jeweiligen Methoden leisten sollen.
- ⇒ Beschreibung für jede Methode ist notwendig
- ⇒ Beschreibung kann auch direkt als Java-Kommentar vor jeder Methode geschrieben werden. Nutze speziellen Kommentar:

```
/**  
 * Beschreibung  
 * ...  
 */
```

- *Javadoc* kann nun HTML-Seiten daraus generieren.
- ⇒ Dokumentation für Nutzer und Entwickler der Klassen.
- Spezialbefehle in der Beschreibung für Referenzen, Parameter-Beschreibung, ...

Javadoc

- Schnittstellendokumentationen müssen klar definieren, was die jeweiligen Methoden leisten sollen.
- ⇒ Beschreibung für jede Methode ist notwendig
- ⇒ Beschreibung kann auch direkt als Java-Kommentar vor jeder Methode geschrieben werden. Nutze speziellen Kommentar:

```
/**  
 * Beschreibung  
 * ...  
 */
```

- *Javadoc* kann nun HTML-Seiten daraus generieren.
- ⇒ Dokumentation für Nutzer und Entwickler der Klassen.
- Spezialbefehle in der Beschreibung für Referenzen, Parameter-Beschreibung, ...

Javadoc

- Schnittstellendokumentationen müssen klar definieren, was die jeweiligen Methoden leisten sollen.
- ⇒ Beschreibung für jede Methode ist notwendig
- ⇒ Beschreibung kann auch direkt als Java-Kommentar vor jeder Methode geschrieben werden. Nutze speziellen Kommentar:

```
/**  
 * Beschreibung  
 * ...  
 */
```

- *Javadoc* kann nun HTML-Seiten daraus generieren.
- ⇒ Dokumentation für Nutzer und Entwickler der Klassen.
- Spezialbefehle in der Beschreibung für Referenzen, Parameter-Beschreibung, ...