Überblick

- Methoden? Na, klar!
- Parameterübergabe und Referenzen
- Das Schlüsselwort `static`

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

Parameterübergabe und Referenzen

Parameterübergabe und Referenzen

```
Typ x;
void method(Typ y) {
  if (...) y = something;    (1)
  if (...) y.modify();       (2a)
  if (...) y[4] = 8;         (2b)
}
...
method(x);
```

Was passiert mit x in (1), (2a), (2b)?

Parameterübergabe und Referenzen

```
Typ x;
void method(Typ y) {
  if (...) y = something;    (1)
  if (...) y.modify();       (2a)
  if (...) y[4] = 8;         (2b)
}
...
method(x);
```

- Java kennt nur Wertparameter

⇒ Variable y enthält Kopie des übergebenen Wertes x

⇒ Falls Typ Basisdatentypen, kann x nicht beeinflusst werden

- Der Wert eines Objekts oder eines Arrays ist eine Referenz

⇒ Falls Typ ein Objekt oder Array ist, referenzieren x und y das gleiche Objekt. Änderungen des Objekts beeinflussen x.

Parameterübergabe und Referenzen

```
Typ x;
void method(Typ y) {
  if (...) y = something;   (1)
  if (...) y.modify();      (2a)
  if (...) y[4] = 8;        (2b)
}
...
method(x);
```

- Änderungen vom Typ (1) weisen y neuen Wert zu

$\Rightarrow$ kein Einfluss auf x

$\Rightarrow$ falls y Referenz auf anderes Objekt enthält, beeinflussen nachfolgende Änderungen an dem von y referenzierten Objekt nicht das von x referenzierte Objekt

Parameterübergabe und Referenzen

```
Typ x;
void method(Typ y) {
  if (...) y = something;    (1)
  if (...) y.modify();       (2a)
  if (...) y[4] = 8;         (2b)
}
...
method(x);
```

- Änderungen vom Typ (2) ändern das von y referenzierte Objekt

$\Rightarrow$ Änderungen beeinflussen auch x (Seiteneffekt)

- Seiteneffekte nicht direkt sichtbar, oft Fehlerquelle

Das Schlüsselwort `static`

Nicht-statische Attribute

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

- jedes Objekt der Klasse hat seine eigenen Attribute

$\Rightarrow$ Zugriff auf Attribut nur über Objekte möglich

$\Rightarrow$ Änderungen betreffen nur Attribute des Objekts
und nicht andere Objekte der Klasse
(beachte jedoch Referenzen)

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

Nicht-statische Methoden (`someMethod`)

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

- Aufruf nur über Objekte möglich
- Methode hat Zugriff auf `this`

$\Rightarrow$ direkter Zugriff auf alle Attribute der Klasse

$\Rightarrow$ direkter Zugriff auf alle Methoden der Klasse

Statische Methoden (someMethod)

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

- Aufruf über Klasse möglich

$\Rightarrow$ benötigt keine Instanz der Klasse zum Aufruf

- `this` ist im Methodenrumpf nicht zulässig

$\Rightarrow$ direkter Zugriff nur auf statische Attribute der Klasse

$\Rightarrow$ direkter Zugriff nur auf statische Methoden der Klasse