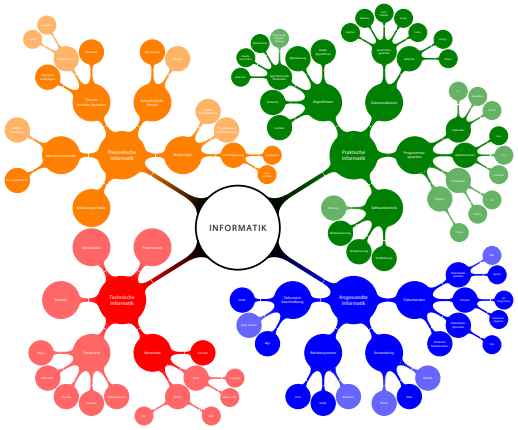
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 15

Modularisierung im Kleinen

Wie man große Probleme in genießbare Häppchen aufteilt

Vorlesung Einführung in die Informatik 1 vom 12. Dezember 2013 von Till Tantau



Lernziele von Kapitel 15

1. Das Konzept des Unterprogramms/Methode verstehen und anwenden können
2. Java-Syntax der Modularisierung auf Methodenebene beherrschen

Gliederung von Kapitel 15

Wie trimmt man einen String von hinten?

Problemstellung

In einem String sollen alle Leerzeichen am Ende entfernt werden.

Lösungsidee

- Es sind bereits Algorithmen zum Umdrehen und zum Trimmen vorne vorhanden.
- Hinten-Trimmen geht dann so:
  1. Drehe den String um.
  2. Trimme vorne.
  3. Drehe den String um.

Zur Diskussion: Welche Probleme hat folgende Lösung?

```
String s = "trim␣me␣␣␣";
{ // first reverse:
  String result = "";
  for (int i = s.length() - 1; i >= 0; i=i-1) {
    result = result + s.charAt(i);
  }
  s = result;
}
{ // trim at front:
  String result = "";
  int start = 0;
  while (start < s.length () && s.charAt(start) == '␣') {
    start = start + 1;
  }
  for (int i = start; i < s.length(); i = i+1) {
    result = result + s.charAt(i);
  }
  s = result;
}
{ // second reverse:
  String result = "";
  for (int i = s.length() - 1; i >= 0; i=i-1) {
    result = result + s.charAt(i);
  }
  s = result;
}
```

Lösung des Problems der Wiederholung.

- Wie brauchen eine Möglichkeit, sich wiederholende Programmteile in ein eigenes *Unterprogramm* auszulagern.
- Solche Unterprogramme haben viele Namen:
  - *Methode* (bei objektorientierten Sprachen wie Java)
  - Funktion (bei funktionalen und imperativen Sprachen)
  - Prozedur (bei imperativen Sprachen)
  - Unterprogramm (bei imperativen Sprachen)

  Es ist immer das gleiche Konzept.

Methoden sind kleine EVAs.

Methoden arbeiten nach dem EVA-Prinzip:

1. Eine Methode bekommt *Eingaben* (aber typischerweise *nicht* von der Tastatur!).
   Beispiel: Ein String
2. Eine Methode *verarbeitet* ihre Eingabe.
   Beispiel: Sie dreht den String um
3. Eine Methode produziert *Ausgaben* (aber typischerweise *nicht* auf dem Bildschirm!).
   Beispiel: Der umgedrehte String

Das Programm mit Methoden

```
String s = "trim␣me␣␣␣";
s = reverse (s);
s = trim (s);
s = reverse (s);
```

Wie die Methoden hingeschrieben werden, kommt gleich.

Die Bestandteile einer Methode in Java.

Eine Methode besteht aus

- einem *Namen* (ein Java-Bezeichner),
- einer Anzahl von (Eingabe)-*Parametern*,
- einem *Körper*,
- einem *Ausgabetyp* und
- *Attributen* wie `public` oder `static`.

Die Attribute interessieren uns erstmal nicht, wir geben aber immer `static` an.

Die prinzipielle Syntax von Methoden in Java.


```
static return_type method_name(parameters)
{
  // Body
  return some_value;
}
```

Es ist

- *return_type* der Typ des Rückgabewertes.
- *method_name* der Name der Methode.
- *parameters* eine Liste der Parameter.
- *some_value* ein Ausdruck vom Typ return_type.

Die Reverse-Methode.

```
static String reverse (String s)
{
  String result = "";

  for (int i = s.length() - 1; i >= 0; i=i-1) {
    result = result + s.charAt(i);
  }

  return result;
}
```

Syntax des Methodennames.

- Der Name einer Methode muss ein *Java-Bezeichner* sein.
- Er darf also keine Leerzeichen oder Sonderzeichen enthalten und muss mit einem Buchstaben anfangen.
- Es ist *üblich*, dass er mit einem Kleinbuchstaben anfängt, und
- es *sollte* ein guter Bezeichner gewählt werden.

Die Reverse-Methode.

```
static String reverse (String s)
{
  String result = "";

  for (int i = s.length() - 1; i >= 0; i=i-1) {
    result = result + s.charAt(i);
  }

  return result;
}
```

Syntax der Parameter.

- Eine Methode kann *mehrere* (Eingabe-)Parameter haben,
- die durch *Kommata* getrennt werden.
- Die Parameter *folgen* dem Methodennamen und stehen in *runden Klammern*.
- Für *jeden* Parameter muss sein Typ angegeben werden, wie bei einer Deklaration.
- In einer Methode kann man Parameter wie andere Variablen auch behandeln. Man *sollte* ihnen aber nichts zuweisen.

Die Reverse-Methode.

```
static String reverse (String s)
{
  String result = "";

  for (int i = s.length() - 1; i >= 0; i=i-1) {
    result = result + s.charAt(i);
  }

  return result;
}
```

Syntax der Parameter.

Methoden ohne Parameter.

- Die Klammern nach dem Methodennamen sind selbst dann nötig, wenn es gar keine Parameter gibt.

```
static double pi ()
{
  return 3.141592653589793;
}
```

5-Minuten-Aufgabe

Zur Übung

Welche der folgenden Parameterlisten sind korrekt?

1. `static int foo (int i)`
2. `static int foo (int i, int j)`
3. `static int foo (int i, j)`
4. `static int foo (int i; int j)`
5. `static int foo (int i, String a, char b)`
6. `static int foo (())`

Syntax des Rückgabetyps.

- Eine Methode kann nur einen *einzigen Wert* zurückliefern. Man sagt auch, die Method *gibt* einen Wert *aus*, was aber nichts mit dem Schreiben auf den Bildschirm zu tun hat.
- Der Typ des Wertes ist der *Rückgabetyp*.
- Dieser wird der ganzen Methode aus historischen Gründen *vorangestellt*.
- Innerhalb einer Methode kann man jederzeit mittels `return` die Methode verlassen und einen Wert zurückliefern.

Die Reverse-Methode.

```
static String reverse (String s)
{
  String result = "";

  for (int i = s.length() - 1; i >= 0; i=i-1) {
    result = result + s.charAt(i);
  }

  return result;
}
```

Syntax des Rückgabetyps.

- Produziert eine Methode überhaupt keinen Rückgabewert, so gibt man statt des Typs ***void*** an.

```
static void printInParantheses (String s)
{
  System.out.println("(" + s + ")");
}
```

5-Minuten-Aufgabe

Zur Übung

Geben Sie den Code einer Methode an, die a^n berechnet. Die Methode arbeitet also wie folgt:

- Die *Eingabe* ist sind zwei Zahlen a und n.
- In der *Verarbeitung* wird $a \cdot a \cdot \ldots \cdot a$ berechnet.
- Das Produkt wird als *Ausgabe* zurückgegeben.

Wie benutzt man nun Methoden?

- Hat man eine Methode deklariert, so kann man sie *aufrufen*.
- Man gibt den Namen der Methode an, gefolgt von Parametern in Klammern (wie in der Mathematik bei $\sin(\pi)$).
- *Innerhalb der Methode* haben die *Parameter* nun die *Werte, die beim Aufruf angegeben* wurden.

Beispiel

- Der *Kopf* von `reverse` lautet:

```
static String reverse (String s)
```

- Zwei mögliche Aufrufe der Methode:

```
... reverse ("Hallo") ...
... reverse (str) ...
```

- Bei ersten Aufruf ist innerhalb von `reverse` die Variable `s` gleich `"Hallo"`. Bei zweiten Aufruf ist `s` gleich `str`.

Das ganze Programm.

```
class Trimmer {
  static String reverse (String s) {
    String result = "";
    for (int i = s.length() - 1; i >= 0; i=i-1) {
      result = result + s.charAt(i);
    }
    return result;
  }
  static String trim (String s) {
    String result = "";
    int start = 0;
    while (   start < s.length ()
           && s.charAt(start) == '␣') {
      start = start + 1;
    }
    for (int i = start; i < s.length(); i = i+1) {
      result = result + s.charAt(i);
    }
    return result;
  }
  public static void main (String[] args) {
    System.out.println
      (reverse(trim(reverse("trimm␣mich␣␣␣"))));
  }
}
```

Zusammenfassung

Syntax einer Methoden-Deklaration

```
static return_type method_name(typ1 parameter1,
                               typ2 parameter2, ...)
{
  // Body
  return some_value;
}
```

- *return_type* ist der Typ des Rückgabewertes.
- *method_name* ist der Name der Methode.
- *typ1* ist der Typ von *parameter1*, entsprechend mit weiteren Parametern.
- *some_value* ist ein Ausdruck vom Typ return_type.

Besonderheiten:

- Gibt es keine Parameter, so müssen trotzdem (leere) runde Klammern stehen.
- Gibt es keine Rückgabewerte, so gibt man als Typ **void** an.

Zusammenfassung

Syntax eines Methoden-Aufrufs

Man ruft eine Methode auf mittels `method(wert1,wert2,...)`.

- Ein solcher Aufruf ist ein *Ausdruck*. Er wertet zum *Rückgabewert der Methode aus*.
- Innerhalb der Method haben die Parameter-Variablen die Werte, die beim Aufruf angegeben wurden.
- Selbst wenn es keine Parameter gibt, müssen trotzdem die runden Klammern geschrieben werden.