Fragen & Wiederholung

CoMa-Übung VIII

TU Berlin

11.12.2012

Themen der Übung

Organisatorisches

Ort und Zeit

- Morgen, 12.12.12, Punkt 14:00 im Audimax (H 105).
- Bringt einen dokumentenechten Stift & euren Studentenausweis mit.

Der Test

8 Aufgaben, 63 Punkte, 31.5 Punkte zum Bestehen notwendig.

- Multiple Choice Java-Fragen.
- Speicherbilder, primitive & Referenzdatentypen.
- Java-Code Auswertung.
- Methode schreiben.
- Methode mit Arrays schreiben.
- Klasse schreiben.
- Compilezeit / Laufzeitfehler finden.
- Kürzeste Wege.

Präinkrement & Postinkrement

`for`-Schleife mit Präinkrement

```
1 int prod = 1;
2 int n = 4;
3 for (int i = 0; i < 5; ++i) {
4   prod *= (i % 2 == 0 ? 3 : 1);
5 }
```

- Äquivalent zu:

```
1 int prod = 1;
2 int n = 4;
3 {
4   int i = 0;
5   while (i < 5) {
6     prod *= (i % 2 == 0 ? 3 : 1);
7     ++i;
8   }
9 }
```

Präinkrement & Postinkrement

for-Schleife mit Präinkrement

```
int prod = 1;
int n = 4;
for (int i = 0; i < 5; ++i) {
  prod *= (i % 2 == 0 ? 3 : 1);
}
```

- Präinkrement erhöht den Wert einer Variable um 1 und gibt ihn zurück.
- Postinkrement gibt den Wert einer Variable zurück und erhöht ihn dann um 1.
- Passiert in dem Inkrement-Ausdruck sonst nichts, sind beide identisch.
- Postinkrement ist etwas langsamer als Präinkrement.
- Gilt analog für Prä- und Postdekrement.

Präinkrement & Postinkrement

for-Schleife mit Präinkrement

```
int prod = 1;
int n = 4;
for (int i = 0; i < 5; ++i) {
  prod *= (i % 2 == 0 ? 3 : 1);
}
```

- Wert von `prod` am Ende: $3 \cdot 1 \cdot 3 \cdot 1 \cdot 3 = 27$.

Postdekrement und Zuweisungen

```
int myInt1 = 130;
int myInt2 = myInt1--*2;
```

```
int myInt1 = 130;
int myInt2 = 2*myInt1--;
```

- Wert von `myInt1` am Ende: 129. Wert von `myInt2` am Ende: 260.

Präinkrement & Postinkrement

for-Schleife mit Präinkrement

```
int prod = 1;
int n = 4;
for (int i = 0; i < 5; ++i) {
  prod *= (i % 2 == 0 ? 3 : 1);
}
```

- Wert von `prod` am Ende: $3 \cdot 1 \cdot 3 \cdot 1 \cdot 3 = 27$.

Postdekrement und Zuweisungen

```
int myInt1 = 130;
int myInt2 = --myInt1*2;
```

```
int myInt1 = 130;
int myInt2 = 2*--myInt1;
```

- Wert von `myInt1` am Ende: 129. Wert von `myInt2` am Ende: 258.

Präinkrement & Postinkrement

for-Schleife mit Präinkrement (2)

```
int prod = 1;
for (int k = 0; k < 5; ++k) {
  if (k > 1) {
    if (k <= 3) {
      prod *= k;
    } else {
      prod *= k + 1;
    }
  }
}
```

- Wert von `prod` am Ende: $2 \cdot 3 \cdot 5 = 30$.

Präinkrement & Postinkrement

for-Schleife mit Präinkrement (3)

```
int x = 10;
int i = 3;
do {
  --i;
} while (--i > 0);
x *= i;
```

- Wert von x am Ende: −10.

Compilerfehler

Compilerfehler

```
float f = 10.3;
int i = 0;
float ff = f + i;
```

- 10.3 ist ein double, der einem float zugewiesen wird.

Compilerfehler (2)

```
public int maximum(int a, int b) {
  if (a < b) return b;
  if (a >= b) return a;
}
```

- Aus Sicht des Compilers ist der return nicht garantiert.

Compilerfehler

Compilerfehler

```
int d = 2;
while (d <= 10) do {
  d++;
}
```

- while - do gibt es nicht.

Scope

Scope bezeichnet die Region eines Programms, in der auf etwas direkt zugegriffen werden kann (→ insbesondere für Variablen wichtig).

- { } leiten einen neuen Scope ein.
- Variablen sind in dem kleinsten umschließenden Scope gültig.

Gültigkeit von Variablen

Arten der Variablen-Deklaration

Variablen können in Java an drei Stellen deklariert werden:

- Als *Attribut* einer Klasse,
- als *Parameter* einer Methode,
- oder als *lokale Variable* in einer Methode.

```
public class Scope {

  private String name; // Attribut

  public void test(int a) { // Methoden-Parameter
    int b = 0; // lokale Variable
    for (int i = 0; i < 10; i++) { // lokale Variable
      int c = 0; // lokale Variable
    }
  }
}
```

Gültigkeit von Variablen (2)

Gültigkeit

- *Attribute* in der ganzen Klasse gültig.
- *Parameter* einer Methode sind in der ganzen Methode gültig (und nirgendwo sonst in der Klasse).
- *Lokale Variablen* sind in dem Block ({..}) gültig, in dem sie deklariert wurden (und nirgendwo außerhalb).

```
1  public class Scope {
2    public void test(int a) {
3      int b = 0;
4      for (int i = 0; i < 10; i++) {
5        int c = 0;
6        System.out.println(b + c); // okay
7      }
8      System.out.println(c); // Fehler
9    }
10 }
```


Gültigkeit von Variablen (3)

Eindeutigkeit von Variablennamen

- An keiner Stelle des Codes dürfen zwei Variablen mit gleichem Namen gültig sein.
- Einzige Ausnahme: *Parameter* / *lokale Variablen* dürfen denselben Namen wie ein *Attribut* haben.
- Java geht dann zunächst davon aus, dass der Parameter bzw. die lokale Variable gemeint ist.
- → Fehlergefahr, nach Möglichkeit vermeiden.

```
1  public class Scope {
2    String a;
3    public void test(int a, int b) { // okay
4      int b = 0; // Fehler
5    }
6  }
```


`switch`

Der `switch`-Befehl

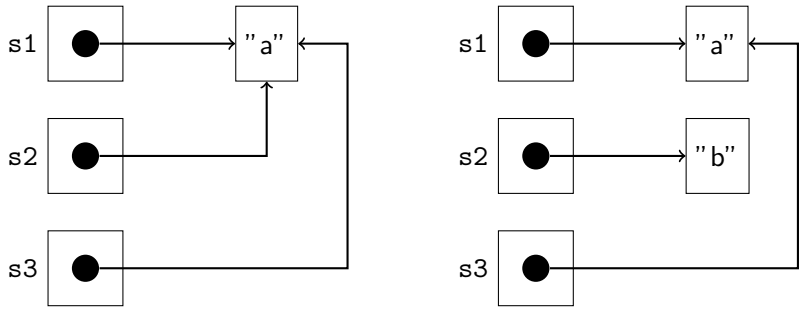
- Für Fallunterscheidungen von Ganzzahlen, Zeichen, Zeichenketten oder Aufzählungstypen mit vielen Fällen.
- Kurzschreibweise um `if`-Befehle mit vielen `else if`s zu vermeiden.
- Kann durch `if`-Befehle simuliert werden.

```
switch (month) {
 case 1:  System.out.println("Januar"); break;
 case 2:  System.out.println("Februar"); break;
 default:  System.out.println("Kein anderer Fall passt.");
}
```

- Nur einzelne Konstanten als Fälle möglich (keine Wertebereiche).
- Switch springt zum ersten passenden Fall und arbeitet den Fall und alle folgenden ab, bis zum Ende des `switch` oder `break`/`return`.
- `default:` ist ein spezieller Fall, der immer passt.

Referenzen und Speicherbilder

```
1  double x = 1.0;
2  double y = 2.0;
3  String s1 = "a";
4  String s2 = s1;
5  String s3 = s2;
6  s2 = "b";
```



Kürzeste Wege

Sei $G = (V, A)$ ein gerichteter Graph mit $n := |V|$ und Entfernungsmatrix A.

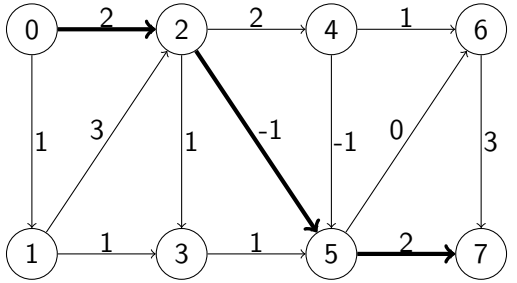
Bellman-Gleichungen

Die **Bellman-Gleichungen** sind gegeben durch:

$$u_{ij}^{(1)} = a_{ij} \quad \text{für alle } i, j \in \{1, \ldots, n\},$$
$$u_{ij}^{(m+1)} = \min_{k=1,\ldots,n} \{u_{ik}^{(m)} + a_{kj}\} \text{ für } m \geq 1 \text{ und } i, j \in \{1, \ldots, n\}.$$

- a_{ij}, $i, j \in \{1, \ldots, n\}$ ist der Eintrag in der Entfernungsmatrix A in der i-ten Zeile und j-ten Spalte, der die Entfernung von i nach j angibt.
- $u_{ij}^{(m)}$, $i, j \in \{1, \ldots, n\}$, $m \geq 1$ bezeichnet die Länge eines kürzesten Weges von i nach j der maximal m Kanten benutzt. Existiert kein solcher Weg, ist der Wert unendlich.

Kürzeste Wege (2)



- Kürzester Weg von 0 zu 7? $0 \to 2 \to 5 \to 7$
- Hinweis: Tree-Matrix und Kürzeste Wege Rekonstruktion nicht test-relevant.