252-0027

Einführung in die Programmierung

Übungen

Präzedenz und Short-Circuiting

Henrik Pätzold
Departement Informatik
ETH Zürich

Organisatorisches

- Mein Name: Henrik Pätzold
- Bei Fragen: **hpaetzold@ethz.ch**
  - Mails bitte mit «[EProg24]» im Betreff
- Material auf: **henrikpaetzold.de**
- Neue Aufgaben: **Dienstag Abend** (im Normalfall)
- Abgabe der Übungen bis **Dienstag Abend (23:59)** Folgewoche
  - Abgabe immer via Git
  - Lösungen in separatem Projekt auf Git

Wie geht man EProg im zweiten Versuch an

- Übungsaufgaben sollten selbstständig gelöst werden (Internet und Slides sind hilfreich, ChatGPT **in kleinen Mengen** auch)
  - Wenn der Arbeitsaufwand steigt, priorisiere ich prüfungsrelevante Aufgaben.
- Vorlesungs-Slides regelmäßig überfliegen
- Fragen stellen, wenn Ihr sie habt
- EProg sollte nicht geringste Priorität haben

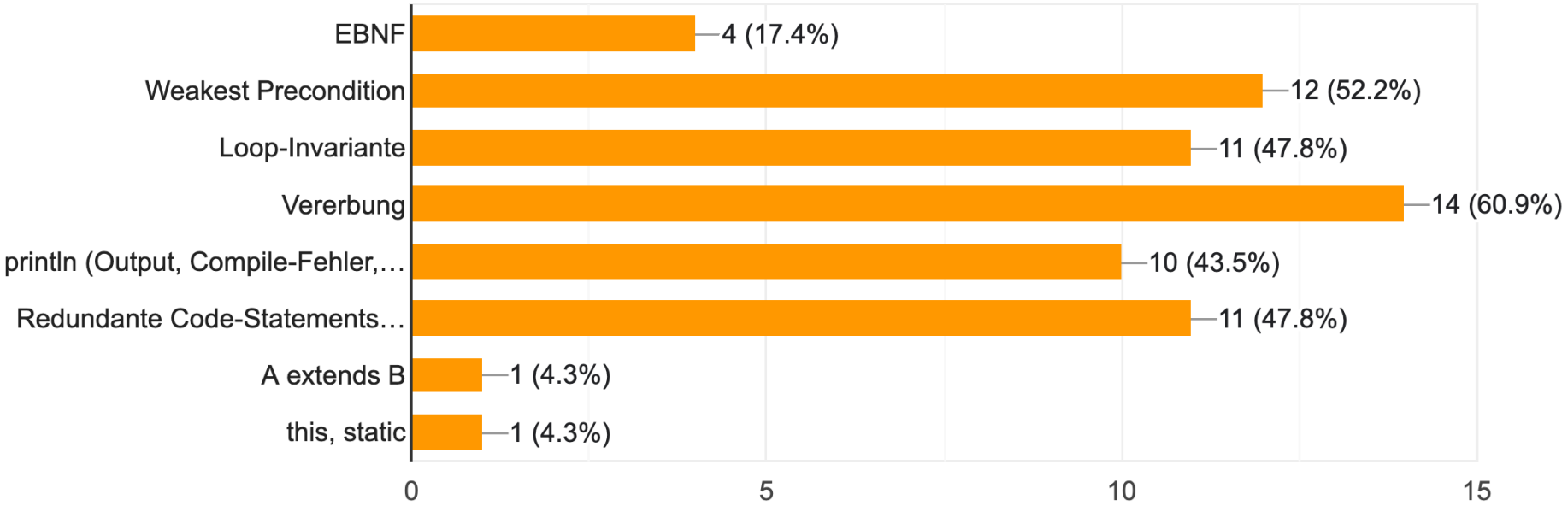
Wichtige Änderung für Repetenten (Nochmal)

- Neue Version: ***Java 21 (Vorher Java 17)***
  - Java 21 aus dem Oracle Archiv installieren -> JRE 21 in Eclipse als Standard auswählen
  - Informationen zu Fehlermeldungen und Behebungen direkt auf der Website
    - Sonst Eclipse und Java deinstallieren, JDK 21 und Eclipse neu installieren
  - Sehr wichtig, weil Korrektur der Bonusaufgaben auch Java 21 benutzt

Mehr bei Eclipse

Diese Themen bereiten mir im Theorie-Teil Mühe

23 responses



Heutiger Plan

- Theorie
  - Println-Statements (Präzedenz)
  - Short circuiting
  - Redundanz
- Vorbesprechung - Übung 2
- Kahoot

Operator-Präzedenz

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operator-Präzedenz

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Wie löse ich diesen Aufgabentypen?

- Nach Präzedenz Klammern setzen
  - Falls zwei Operatoren die gleichen Klammern haben, gilt linksassoziativität: a+b+c = (a+b)+c
- Klammern zuerst auflösen, bevor zwischen größeren Teiltermen gerechnet wird
- Auf den Datentyp zwischen zwei Operationen achten (z. B. String + int = String, int + double = double)

5 + 3 * 2 - 4 / 2 + "" + 6 % 4 + (8 / 2) * 3

$$[(\{[5 + (3 * 2)] - (4 / 2)\} + "") + (6 \% 4)] + ((8 / 2) * 3)$$

Wie Java den Ausdruck klammern würde

5 + 3 * 2 - 4 / 2 + "" + 6 % 4 + (8 / 2) * 3

5 + (3 * 2) – (4 / 2) + "" + (6 % 4) + ((8 / 2) * 3))

Dies ist nicht wie Java das Problem angeht, es ist der minimale Aufwand, so dass man, wenn man sich mit der Präzedenz von Operatoren auskennt, diese Aufgaben schnell an einer Prüfung lösen kann

5 + 6 – 2 + "" + 2 + 12

$$[(\{5 + 6\} - 2) + ""] + 2 + 12$$

[(9) + ""] + 2 + 12

"9"+ 2 + 12

("9"+ 2) + 12

"9212"

Arithmetische Compiler-Fehler und Exceptions

- Compiler-Fehler treten auf, bevor die Laufzeit beginnt
  - Häufigste Compiler-Fehler: Wenn zwei inkompatible Datentypen in einer Operation verwendet werden.
  - *Beispiel*: int x = 5.0 + 2, “hallo”*3
- Exceptions treten zur Laufzeit auf
  - Häufigste arithmetische Exception: Division durch Null
  - Der Compiler überprüft nur die Syntax, nicht die Funktionsweise des Programms (z. B. logische Fehler werden nicht erkannt).

Aufgabe 1

```java
class MyClass{
    public static void main(String[] args){
        System.out.println(9 / 3 * 2 + 1 + "" + 2 * 3 + (4 * 3) % 3);
    }
}
```

"760"

Aufgabe 2

```java
class MyClass{
    public static void main(String[] args){
        System.out.println(8 / 5 + 0.5 + 5 / 2 + (8 % 3) * 1.0);
    }
}
```

5.5

Präzedenz & Short Circuiting

Wie löse ich diesen Aufgabentypen?

- && (Logisches UND): Wenn der erste Ausdruck false ist, wird der zweite nicht ausgewertet.
  - *Beispiel:* x>0 && 1/x==Integer.MAX_VALUE
- || (Logisches ODER): Wenn der erste Ausdruck true ist, wird der zweite nicht ausgewertet.
  - *Beispiel:* x == 0 || 1/0==Integer.MAX_VALUE

Aufgabe 1

```java
class MyClass{
    public static void main(String[] args){
        System.out.println(7 > 6 && (3 + ">" + 2) == "3 > 2" || 4 % 2 == 0 % 0);
    }
}
```

Exception (Division by Zero)

Aufgabe 2

```java
class MyClass{
    public static void main(String[] args){
        int a = 0;
        int b = 5;
        System.out.println((a > 0) && (b / a > 1)); // Was wird ausgegeben? Warum?
    }
}
```

False

Aufgabe 3

```java
class MyClass{
    public static void main(String[] args){
        int x = 3;
        int y = 10;
        System.out.println((x < y) || (x > 5) && (y++ == 10)); // Was wird ausgegeben? Was ist der Wert von y?
    }
}
```

True, y=10

Aufgabe 4

```java
class MyClass{
    public static void main(String[] args){
        int a = 3;
        int b = 4;
        System.out.println((a++ == 3) && (++b == 5)); // Was wird ausgegeben? Was sind die Werte von a und b nach dem Aufruf?
    }
}
```

True, a=4, b=5

Aufgabe 5

```java
class MyClass{
    public static void main(String[] args){
        int x = 2;
        int y = 8;
        System.out.println((x < y || ++x == 3) && (x++ == 3 || y-- > 5)); // Was wird ausgegeben?
    }
}
```

True, x=3, y=7

Aufgabe 6

```java
class MyClass{
    public static void main(String[] args){
        int m = 7;
        int n = 9;
        System.out.println((m == 7) || (n-- == 9) && (++m == 8) || (m == 9)); // Was wird ausgegeben?
    }
}
```

True, m=7, n=9

Redundanz

Wie löse ich diesen Aufgabentypen?

- Gibt es ungenutzte Variablen?
- Überflüssige Kontrollstrukturen:
  - Wird eine if/else branch immer oder nie ausgeführt? Bedingungen überprüfen!
  - Analog für Schleifen
- Doppelte Statements und Objektverweise

```java
    static int[] staticArr = {1, 2, 3}; // Statische Referenz

    static int sum() {
        int total = 0;
        int[] localArr = staticArr;

        for (int i = 0; i < localArr.length; i++) {
            total += localArr[i];
        }


        localArr[0] = 100;

        if (staticArr == localArr) {
            return total;
        }

        return 0;
    }
}

```

```
// add  the elements of an n-element array

// x is never null
                                                    // Platz fuer Antwort
static double func(double[] x,  int n)
{

    double result = 0.0;                            // Nicht Redundant

    int size = 0;                                   // Redundant

    int elems = x.length;                           // Redundant

    double adjust = 0.0;                            // Nicht Redundant

    if (n != x.length)                              // Nicht Redundant
    {
        adjust++;                                   // Nicht Redundant

        throw new IllegalArgumentException("Size mismatch " + adjust); //
    }

    adjust = 0.0;                                   // Redundant

    int count = x.length;                           // Redundant

    for (int i = 0; i < n; )                        // Nicht Redundant
    {
        count = count - 1;                          // Redundant

        result += x[i];                             // Nicht Redundant

        i++;                                        // Nicht Redundant
    }

    if (n == x.length)                              // Redundant
    {
        double toBeReturned;                        // Redundant

        toBeReturned = result;                      // Redundant

        result += adjust;                           // Redundant

        return result;                              // Nicht Redundant
    }
    return 0.0;                                     // Redundant
}
```

Vorbesprechung Übung 2

Kahoot

Zusatzaufgaben

- Erstellen Sie eine Beschreibung <palindrome>, welche als legale Symbole alle Zahlen zulässt, die von vorne und hinten gleich gelesen werden und die nur die Ziffern von 1 bis 4 verwenden. Beispiele sind 11, 232, 444
- Erstellen Sie eine Beschreibung <five>, welche alle Summen von positiven Zahlen zulässt, welche 5 ergeben. Beispiele sind “1 + 4”, “2 + 1 + 1 + 1”, “5”
- Erstellen Sie eine Beschreibung <oddEight>, die alle Zahlen zulässt, bei denen die Ziffer 8 eine ungerade Anzahl von Malen vorkommt.