# Praktikum: Grundlagen der Programmierung

Prof. Dr. Alexander Pretschner, J. Kranz, G. Hagerer

WS 2018/19

**Übungsblatt 3**

Abgabefrist: s. Moodle

Σ Punkte: 20

### Aufgabe 3.1 (P) Auswertung von Ausdrücken

Geben Sie die Ergebnisse der folgenden Ausdrücke und Zuweisungen an. Lösen Sie die Aufgabe ohne den Code zu kompilieren und auszuführen.

**a)**
- Was bedeuten die Begriffe *Ausdruck* und *Statement* ? Worin liegen die Unterschiede? Welche Arten von Statements gibt es? Vgl. hierzu auch `http://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html`
- Was bedeutet *Wert* bzw. *Typ* eines Ausdrucks? Machen Sie sich klar, dass jeder Ausdruck einen eindeutigen Typ hat und zu einem eindeutigen Wert evaluiert werden kann!
- Gibt es auch geschachtelte Ausdrücke? Wie kann man Typ und Wert bei solchen Ausdrücken bestimmen?

**b)** Betrachten Sie das folgende Java-Programm. Was sind Statements und was sind Ausdrücke bzw. Teilausdrücke? Bestimmen Sie außerdem für jede Variable `resXX` deren Wert.

```
1  byte by = 1;
2  int i = 2;
3  long l = 5; // Achtung: 1 != l
4  double d = 7.0;
5  boolean b = true;
6  char c = 'c';
7  String st = "|";
8
9  int res1 = by + i;
10 long res2 = i + l;
11 double res3 = i + l;
12 int res4 = c + 1;
13 long res5 = l % i;
14 long res6 = l / i;
15 double res7 = l / i;
16 double res8 = l / 7;
17 double res9 = l / d;
18 String res10 = c + st;
19 String res11 = st + l;
20 String res12 = b + st;
```

**c)** Was ergibt sich jeweils für `d`?

```java
boolean a = true;
boolean b = false;
boolean c = true;
boolean d;
d = !a;
d = a && b;
d = !a || !c;
d = (a && b) || !c;
d = !(!a && b) || !c;
```

### Aufgabe 3.2 (P) Übergabe von Referenzen

Welche Werte ergeben sich jeweils für die Ausgaben von `"Referenzwert ..."` für das folgende Java-Programm?

```java
public class Referenz {
  int wert;

  public static void main(String[] args) {
    Referenz referenz = new Referenz();
    setzeWert(referenz, 5);
    System.out.println("Referenzwert 1: " + referenz.wert);

    referenz = setzeWert(referenz, 10);
    System.out.println("Referenzwert 2: " + referenz.wert);

    setzeWert(null, 15);
    System.out.println("Referenzwert 3: " + referenz.wert);

    referenz = setzeWert(null, 20);
    System.out.println("Referenzwert 4: " + referenz.wert);

  }

  public static Referenz setzeWert(Referenz referenz, int wert) {
    if (referenz == null) {
      referenz = new Referenz();
    }
    referenz.wert = wert;

    return referenz;
  }
}
```

Machen Sie sich in dieser Aufgabe das Prinzip *call-by-value* deutlich! Welche Auswirkung hat dies auf Parameter welche als Basistyp oder Referenztyp übergeben werden?

### Aufgabe 3.3 (P) Geheimnis- bzw. Lokalitätsprinzip, Datenkapselung

Das Geheimnis- bzw. Lokalitätsprinzip besagt, dass die konkrete Implementierung einer Klasse nach außen hin verborgen wird. Änderungen an der Implementierung einer Klasse haben somit nach außen keine Auswirkung. Dies wird auch **Datenkapselung** genannt. Markieren Sie daher alle Attribute der Klassen `Motor`, `Radar`, `Auto` und `Zeitspanne` aus dem vorigen Aufgabenblatt mit dem Modifier `private`. Somit kann auf die entsprechenden Attribute zunächst nur noch innerhalb der jeweiligen Klasse zugegriffen werden. Schreiben Sie außerdem sogenannte *getter*- und *setter*-Methoden, die mit dem Modifier `public` versehen werden und somit den lesenden und schreibenden Zugriff auf die Attribute von außerhalb der Klasse ermöglichen. Zur Bearbeitung dieser Aufgabe können Sie die bereitgestellte Musterlösung zu den Präsenzaufgaben des vorigen Übungsblattes von der PGdP-Webseite nutzen.

### Aufgabe 3.4 (P) Unit Testing

Für diese Aufgabe testen wir die wie folgt definierte Methode.

```
1  int formula(int x) {
2    return 5 * (int)Math.sqrt(x) / x;
3  }
```

Testen bedeutet, dass wir die entsprechende Methode mit konkreten Parameterwerten aufrufen und die Ergebnisse dieser Methodenaufrufe auf ihre zu erwartenden Werte hin überprüfen. Ein Beispiel für einen solchen Tests bzw. Testfalls ist die folgende Testmethode:

```
1  void test1() {
2    int result = formula(1);
3    if (result == 5)
4      System.out.println("test1() passed. 5 == formula(1)");
5    else
6      System.out.println("test1() FAILED!!! 5 != formula(1)");
7  }
```

Implementieren Sie obige Methode sowie mehrere unterschiedliche Testfälle als Teil einer Testklasse selbst. Beim Testen ist es besonders wichtig, *Sonderfälle* zu betrachten. Bei einer Methode, die eine Zahl als Parameter erwartet, ist z.B. die 0 ein solcher Sonderfall. Schreiben Sie eine Testmethode, die bei Übergabe der 0 erwartet, dass `formula(0)` selbst 0 zurückliefert und diskutieren Sie das Ergebnis!

Überlegen Sie sich außerdem weitere Tests und testen Sie die Methode mit diesen. Testen Sie insbesondere sämtliche Sonderfälle, z.B. Grenzwerte für den zu übergebenden Parameter, an denen die aufgerufene Methode ein anderes Verhalten aufweist als jenseits bzw. diesseits dieses Wertes. Die Testmethoden können aus einer dazugehörigen `main()`-Methode heraus aufgerufen werden.

**Aufgabe 3.5 (P) Binnenmajuskel**

In dieser Aufgabe betrachten wir Strings, die nur aus Kleinbuchstaben ('a' bis 'z'), Großbuchstaben ('A' bis 'Z') und Unterstrichen bestehen.

Gegeben sind die folgenden Regeln, um aus mehreren ($\geq 1$) Begriffen (z. B. Hand, Ball und Tor) entsprechend zusammengesetzte Namen (z. B. Handballtor) zu bilden:

**Startcase** Das Wort beginnt mit einem Großbuchstaben und enthält ansonsten nur Kleinbuchstaben. **Beispiel**: `Handballtor`

**UPPERCASE** Das Wort besteht nur aus Großbuchstaben. **Beispiel**: `HANDBALLTOR`

**snake_case** Die einzelnen Begriffe werden klein geschrieben und durch Unterstriche getrennt. **Beispiel**: `hand_ball_tor`

**PascalCase** Die einzelnen Begriffe werden zusammengeschrieben, jedoch durch Großschreiben des jeweils ersten Buchstabens jedes Begriffs von einander „getrennt". **Beispiel**: `HandBallTor`

Schreiben Sie ein Programm, das es der Benutzerin erlaubt, nacheinander beliebig viele Begriffe einzugeben, die nur aus Buchstaben bestehen. Verwenden Sie dafür die Methoden der mitgelieferten Klasse `Terminal`. Die Groß- bzw. Kleinschreibung der Begriffe darf dabei beliebig *falsch* sein (s. u. das Beispiel). Wird ein ungültiger String eingegeben, soll der Benutzer auf seinen Fehler hingewiesen und die letzte Eingabe ignoriert werden. Die Eingabe der Begriffe soll schließlich durch Eingabe eines leeren Strings beendet werden. Als Ausgabe liefert das Programm dann für jede der oben genannten Konventionen (Startcase, UPPERCASE, snake_case, PascalCase) die entsprechenden Namen. Die jeweilige Zuordnung soll aus der Ausgabe klar hervorgehen.

Beispiel (ohne ungültige Eingaben):

| | |
|---|---|
| Eingabe: | „hand", „BAll", „toR", „". |
| Ausgabe: | „Startcase: Handballtor",<br>„UPPERCASE: HANDBALLTOR",<br>„snake_case: hand_ball_tor",<br>„PascalCase: HandBallTor". |

Vergessen Sie nicht, Ihre Implementierung wie in der vorhergehenden P-Aufgabe beschrieben zu testen. Diskutieren Sie dabei, wie dies trotz Benutzerinteraktion möglich ist!

**Hinweis**: Ihre IDE erlaubt es Ihnen, Ihren Code zu *debuggen*. Dies bedeutet, dass Ihr Code Schritt für Schritt ausgeführt wird, wobei Sie die Ausführung nach jeder Anweisung unterbrechen und sich aktuelle Variablenwerte anzeigen lassen können. Sie können das Debugging starten, indem Sie in Eclipse auf den kleinen Käfer klicken. Debuggen Sie diese Aufgabe zusammen mit Ihrem Tutor!

---

Die Hausaufgabenabgabe erfolgt über Moodle. Bitte geben Sie Ihren Code als UTF8-kodierte (ohne BOM) Textdatei(en) mit der Dateiendung `.java` ab. Geben Sie **keine** Projektdateien Ihrer Entwicklungsumgebung ab. Nutzen Sie keine Packages. Geben Sie **keinen** kompilierten Code ab (`.class`-Dateien). Geben Sie Ihren Code **nicht** als Archiv (z.B. als `.zip`-Datei) ab. Achten Sie darauf, dass Ihr Code kompiliert. Hausaufgaben, die sich nicht im vorgegebenen Format befinden, werden nur mit Punktabzug oder gar nicht bewertet.

### Aufgabe 3.6 (H) *setter*-Methoden [2 Punkte]

In der Hausaufgabe 2.5 haben Sie bereits *getter*-Methoden für die Klassen `Date`, `Document`, `Author` und `Review` geschrieben. Schreiben Sie nun für diese Klassen *setter*-Methoden, mit denen das jeweilige Attribut auf den gegebenen Wert gesetzt werden kann. Beachten Sie beim Schreiben der *setter*-Methoden, dass

- diese dafür verantwortlich sind, dass die Attribute nur auf semantisch sinnvolle Werte gesetzt werden
- das Verhalten der Methode bei Angabe nicht sinnvoller Werte zu dokumentieren ist.
- **Beispiel: Datum**. Für Datumsangaben ist es semantisch sinnvoll, dass Tag, und Monat nur bestimmte Werteausprägungen haben: $Tag \in \{1..31\}, Monat \in \{1..12\}$

Stellen Sie im Rahmen dieser Aufgabe außerdem sicher, dass die Attribute auch mittels der Konstruktoren nur auf sinnvolle Werte gesetzt werden können. Halten Sie sich bei der Vergabe der Namen für *setter*-Methoden an die Konvention aus Aufgabe 2.5.

Beachten Sie folgende **Hinweise**:

- Auf dem letzten Blatt haben wir Methoden geschrieben, die Berechnungen auf Basis von 30 Tagen pro Monat anstellen. Diese Methoden sollen hier nicht angepasst werden.
- Der 29.2.2018 ist kein korrektes Datum, der 29.2.2020 dagegen schon.
- Eine korrekte Mail-Adresse enthält ein @-Symbol. Ansonsten gibt es keine Anforderungen.

### Aufgabe 3.7 (H) Die Klasse `WordCount` [5 Punkte]

Was für die geplante Suchmaschine bisher fehlt, ist der eigentliche Inhalt der Dokumente. In dieser Aufgabe entsteht daher die Klasse `WordCount`, die ein Wort und dessen absolute Häufigkeit innerhalb eines Dokumentes repräsentiert. Die Häufigkeit eines Wortes wird später die grundlegende Information für die Bewertung der Dokumente durch die Suchmaschine sein.

1. Die Klasse `WordCount` repräsentiert jeweils ein Wort (in Form eines `String`s) und dessen Häufigkeit. Fügen Sie der Klasse `WordCount` daher entsprechende `private`-Attribute hinzu.
2. Stellen Sie außerdem einen Konstruktor sowie *getter*-Methoden für die Attribute zur Verfügung.
3. Eine *setter*-Methode ergibt dagegen nur für die Häufigkeit des Wortes Sinn. Das durch diese Klasse repräsentierte Wort soll nachträglich nicht verändert werden können. Schreiben Sie daher eine *setter*-Methode für die Häufigkeit des Wortes.
4. Schreiben Sie außerdem die Methoden

```
public int incrementCount()
```

und

```
public int incrementCount(int n),
```

bei deren Aufruf die Häufigkeit um `1` bzw. um `n` erhöht wird. Zusätzlich soll bei beiden Methoden der neue Wert der Häufigkeit zurückgegeben werden. Beachten Sie, dass dabei für `n` ein negativer Wert übergeben werden kann. Die Häufigkeit des Wortes soll in diesem Fall nicht verändert werden.

5. Testen Sie alle öffentlichen Methoden der Klasse `WordCount`. Implementieren Sie dafür jeweils mehrere Testmethoden, welche insbesondere alle Sonderfälle und mehrere sonstige Fälle überprüfen. Argumentieren Sie in einem Kommentar kurz, wieso Sie den jeweiligen Testfall gewählt haben.

Achten Sie bei den Teilaufgaben 2, 3 und 4 darauf, dass illegale Werte übergeben werden können. Identifizieren Sie diese Fälle und behandeln Sie diese angemessen. Beachten Sie, dass die Art und Weise der Behandlung dieser Fälle für den Anwender der Klasse dokumentiert werden muss.

### Aufgabe 3.8 (H) Primverquerung [5 Punkte]

Schreiben Sie die Methode

```
public static int querPrim(int n),
```

der Klasse `Primverquerung`, die für eine Zahl `n` die Summe derjenigen Primzahlen $p < n$ berechnet, deren Quersumme gerade ist. Die Methode soll bei Eingabe negativer Zahlen den Wert 0 zurückgeben.

Testen Sie Ihre Implementierung in einer zusätzlichen `main()`-Methode!

**Hinweis**: Verwenden Sie keine Klassen oder Methoden der Java-Standardbibliothek.

### Aufgabe 3.9 (H) Stringulina [8 Punkte]

Erstellen Sie die Klasse `Stringulina` und implementieren Sie die folgenden statischen Methoden in dieser Klasse.

- Implementieren Sie die Methode

  ```
  public static int substringPos(String haystack, String needle),
  ```

  die einen String `needle` innerhalb eines Strings `haystack` sucht und seine erstmögliche Anfangsposition zurückliefert. Wird `needle` nicht gefunden, soll $-1$ zurückgegeben werden.

- Implementieren Sie die Methode

```
public static int countSubstring(String haystack, String needle),
```

die zählt, wie häufig der String `needle` in einem zweiten String `haystack` vorkommt. **Hinweis**: Der Substring `"aa"` kommt innerhalb von `"aaaa"` 3 Mal vor.

- Implementieren Sie die Methode

  ```
  public static boolean correctlyBracketed(String str),
  ```

  die überprüft, ob der gegebene String `str` in Bezug auf runde Klammern korrekt geklammert ist. Ein String ist korrekt geklammert, wenn auf jede öffnende Klammer eine schließende Klammer folgt und jede schließende Klammer einer öffnenden Klammer zugeordnet werden kann. Ist `str` korrekt geklammert, so soll `true` zurückgegeben werden, ansonsten `false`.

  Beachten Sie folgende Beispiele:

  - `"a(xx(]))"` ist korrekt geklammert.
  - `"a(xx))"` ist nicht korrekt geklammert.
  - `"a(xx)("` ist nicht korrekt geklammert.
  - `"a)xx()("` ist nicht korrekt geklammert.

- Implementieren Sie die Methode

  ```
  public static boolean matches(String str, String pattern),
  ```

  die prüft, ob ein String `str` bestehend aus kleinen und großen Buchstaben (a-z und A-Z) zu dem gegebenen Pattern `pattern` passt. Ein Pattern kann neben Buchstaben zwei besondere Konstrukte enthalten:

  1. Ein Punkt (`'.'`) passt zu, d.h. "matcht", jeden Buchstaben.
  2. Hinter einem Buchstaben oder einem Punkt kann in geschleiften Klammern ({, }) eine Vielfachheit angegeben werden.

  Betrachten Sie folgende Beispiele:

  - Das Pattern `"P.{2}ngui{1}."` passt z.B. zu dem String `"Pijnguin"`.
  - Das Pattern `"Ha{10}..o"` passt z.B. zu dem String `"Haaaaaaaaaawko"`, aber nicht zu `"Haaawko"`.

  **Hinweis**: Fehlerbehandlung ist nicht verlangt. Sie können also davon ausgehen, dass sowohl `str`, als auch das Pattern valide sind.

Testen Sie Ihre Implementierung. Erstellen Sie dazu eine Klasse `StringulinaTest`, welche Testfälle für alle Methdoen der Klasse `Stringulina` implementiert.

**Hinweis**: Sie dürfen im Rahmen dieser Aufgabe lediglich die Methoden `char charAt(int pos)` und `int length()` der Klasse `String` sowie `int Integer.parseInt(String s)` verwenden. Sie dürfen natürlich eigene Methoden schreiben und verwenden.