Fragen, Antworten, Kommentare

Die Online-Befragung zur gewählten alternativen Veranstaltungsform ist online. Bitte ausfüllen: https://forms.gle/xLm21KRaTgs7En7G6

Am 20.12 und 22.12 bin ich nicht zur Vorlesungszeit online, da ja keine Vorlesungen stattfinden. Wir können aber bei Bedarf einen Online-Termin vereinbaren. Natürlich können Sie mir auch in den zwei freien Wochen E-Mails mit Fragen und halbfertigen Projekten schicken.

Das letzte Aufgabenblatt 11, das abgenommen wird, ist online. Die Abnahme ist in der vorletzten Praktikumswoche. Wenn das Blatt abgenommen wurde, besteht keine Verpflichtung mehr zur Praktikumsteilnahme. Die Praktikumsleiter sind natürlich trotzdem erreichbar.

Ein gerne gemachter objektorientierter Fehler wird hier am Beispiel von 28 m) gezeigt, dabei wird eine bereits vorher geschriebene Hilfsmethode vorkommenVon() genutzt, die die Anzahl der Vorkommen eines Wertes zählt.

```
public int vorkommenVon(Messreihe mr, int wert) { // schlecht!!!
  if (mr == null || mr.getMesswerte() == null) {
    return 0;
  }
  int ergebnis = 0;
  for (int i: mr.getMesswerte()) {
    if (i == wert) {
      ergebnis = ergebnis + 1;
    }
  }
  return ergebnis;
}

public boolean gleicheWerte(Messreihe m) {
  if (this.messwerte == null || this.messwerte.isEmpty()) {
    return m == null || m.getMesswerte() == null
        || m.getMesswerte().isEmpty();
  }
  if (m == null || m.getMesswerte() == null) {
    return false;
  }
  for (int i: this.messwerte) {
    if (m.vorkommenVon(m, i) == 0) {
      return false;
    }
  }
  for (int i: m.getMesswerte()) {
    if (this.vorkommenVon(this, i) == 0) {
      return false;
    }
  }
  return true;
}
```

Generell läuft die Lösung, aber der Ansatz der Methode vorkommenVon() ist falsch, da hier als erster Parameter eine Messreihe (genauso schlecht eine ArrayList) übergeben wird. Wir haben damit in der Klasse Messreihe eine Methode geschrieben, die eine andere Messreihe verarbeiten soll. Betrachtet man die Idee von vorkommenVon() aber genauer, bezieht sich der Aufruf immer genau auf die Messreihe, deren Methode aufgerufen wird. Daraus folgt das der Parameter hier überflüssig ist. Aus Sicht der Objektorientierung ist das ein massiver Fehler, den man genau einmal machen darf. Die Korrekturen sehen wie folgt aus.

```
public int vorkommenVon(int wert) {
  if (this.messwerte == null) {
    return 0;
  }
  int ergebnis = 0;
  for (int i: this.messwerte) {
    if (i == wert) {
      ergebnis = ergebnis + 1;
    }
  }
  return ergebnis;
}

public boolean gleicheWerte(Messreihe m) {
  if (this.messwerte == null || this.messwerte.isEmpty()) {
    return m == null || m.getMesswerte() == null
        || m.getMesswerte().isEmpty();
  }
  if (m == null || m.getMesswerte() == null) {
    return false;
  }
  for (int i: this.messwerte) {
    if (m.vorkommenVon(i) == 0) {
      return false;
    }
  }
  for (int i: m.getMesswerte()) {
    if (this.vorkommenVon(i) == 0) {
      return false;
    }
  }
  return true;
}
```

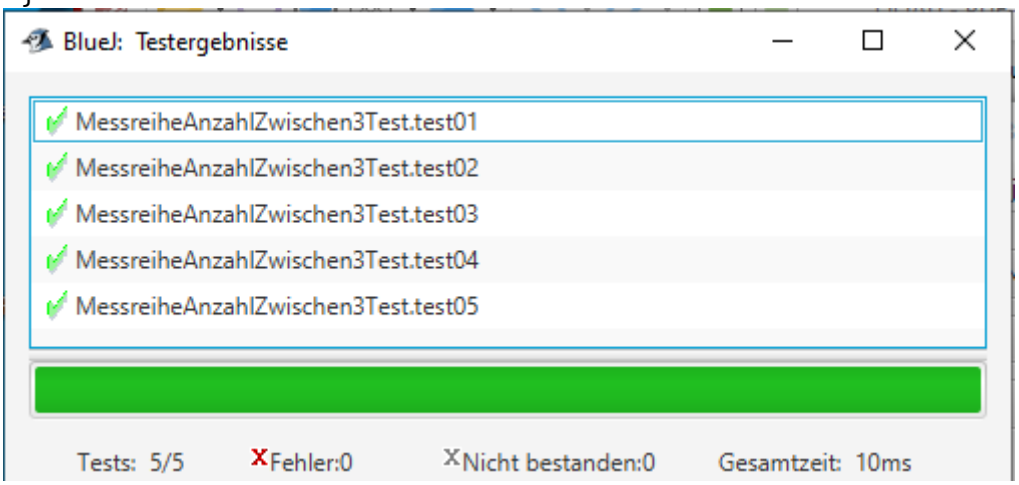
„Do not Copy & Paste“ hat auch mich mal wieder erwischt. In den Tests zur Aufgabe 28 wird in der Testklasse MessreiheAnzahlZwischen4Test.java irrtümlich die Methode anzahlZwischen2() und nicht anzahlZwischen4() getestet. Gleiches gilt für MessreiheAnzahlZwischen3Test.java. Bitte von Hand korrigieren. Damit werden auch die Fehler bei den Studierenden gefunden, die zweimal it.next() aufrufen und den gleichen Wert erwarten. Bei next() wird der aktuelle Wert zurückgegeben und der Iterator ein Element weiter gesetzt. Das wird nebenbei sonst bei Methoden vermieden, dass eine Berechnung stattfindet und dann der Zustand des Objekts, also Werte der Objektvariablen verändert werden. Iteratoren sind da eine absolute Besonderheit.

Erinnerung: Tests helfen bei Entwicklung und ihre Nutzung ist in größeren Projekten ein Standard. Ob Tests vor oder nach der Entwicklung geschrieben werden, hängt wieder von Rahmenbedingungen ab. Generell muss man sich dazu merken, dass Tests notwendig (man muss sie machen) aber nicht

hinreichend für eine gute Software-Qualität sind. Dies bedeutet, dass es trotz vollständig erfüllter Tests immer noch massive Fehler in den Programmen geben kann, siehe auch [Kle19], über die Bibliothek kostenlos herunterladbar. Dies bedeutet für Ihre Praktika, dass Ihre Programme trotz einer recht hohen Testanzahl noch falsch sein können und Sie Ihre Algorithmen weiterhin im Kopf überprüfen müssen. Wenn sowas nebenbei passiert, also falsche Programme mit laufenden Tests, schicken Sie mir solche Programme gerne zu, da ich dann zumindest die Testanzahl erhöhen kann (was nie hinreichend sein wird).

Ein sehr interessantes Beispiel ist diese vermeintliche Lösung zur Aufgabe 28 g)

```
public int anzahlZwischen3(int min, int max) {
  if (this.messwerte == null || this.messwerte.isEmpty()) {
    return 0;
  }
  int ergebnis = 0;
  for (int i : this.messwerte) {
    if (this.messwerte.get(i-1) <= max && this.messwerte.get(i-1) >= min){
      ergebnis = ergebnis + 1;
    }
  }
  return ergebnis;
}
```



Es sollte klar sein, dass hier die ForEach-Schleife nicht richtig verstanden und i nicht als Element, sondern als Index angesehen wurde. Da es mit get(i) eine Exception gibt, wurde mit i-1 experimentiert und siehe da, es klappt auch bei den korrigierten (siehe oben) Tests.

Den Fehler erkennt man mit Programmiererfahrung sehr schnell, aber Sie kommen immer wieder in Situationen, in denen Sie Anfänger auf einem Gebiet sind, so dass ein vergleichbarer Fehler dann auch möglich ist. Deshalb benötigt man z. B. beim Selbststudium immer einen erfahreneren Coach, der zumindest am Ende sich Ergebnisse anschaut.

[Kle19] S. Kleuker, Qualitätssicherung durch Softwaretests, 2. aktualisierte und erweiterte Auflage, Springer Vieweg, Wiesbaden, 2019