Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Dr. Jens Bendisposto, Dr. Markus Brenneis

25. September 2023

Klausur

Programmierpraktikum 1
SS 2023

Hinweise:

- Diese Klausur enthält 9 nummerierte Klausurseiten. Prüfen Sie bitte zuerst, ob die Klausur alle Seiten enthält.
- Täuschungsversuche führen zum sofortigen Ausschluss von der Klausur. Die Klausur wird dann als nicht bestanden gewertet.
- Schreiben Sie **nicht** mit radierbaren Stiften und auch **nicht** mit rot!

□ Falls ich diese Klausur bestehe, möchte ich automatisch zum Programmierpraktikum 2 im kommenden Wintersemester angemeldet werden.

Nachname: _____ Vorname: _____

Matrikelnummer: _____

Hörsaal: _____ Sitzplatznummer: _____

Unterschrift: _____

Viel Erfolg!

Diesen Teil bitte nicht ausfüllen:

Aufgabe	1	2	3	4	5	6	7	Σ
Punktzahl	6	5	4	4	2	1	2	24
Erreicht								

Aufgabe 1 [6 Punkte]

Wir wollen Zahlen zählen, allerdings aufgeteilt nach der letzten Ziffer. Es sollen also z. B. alle Zahlen, deren letzte Ziffer eine 4 ist, getrennt gezählt werden von den Zahlen, deren letzte Ziffer eine 7 ist.

Wir haben eine Instanzmethode `Integer zufall()` in der Klasse `Zahlen` gegeben, die uns eine zufällige, positive Zahl erzeugt.

Nun wollen wir eine Methode `Map<Integer,Long> zaehle(int k)` schreiben, die `k` Zufallszahlen erzeugt und nach der letzten Ziffer getrennt zählt.

Beispiel für `k = 4`: Wenn unser Zufallszahlengenerator die Zahlen `21,1,26,11` erzeugt, betrachten wir also die jeweils letzten Ziffern dieser Zahlen, `1,1,6,1`, und erstellen als Rückgabewert der `zaehle`-Methode `{1=3, 6=1}`.

Ergänzen Sie die Implementierung der Methode `zaehle`. Bei der Berechnung der Map müssen **genau ein Stream** und die Instanzvariable `z` verwendet werden; andere Kontrollstrukturen sind nicht erlaubt. Verwenden Sie immer, wenn es möglich ist, **Methodenreferenzen**. Sie dürfen keine zusätzlichen Methoden schreiben. Die Schlüssel in der Map müssen nicht sortiert sein.

Tipps:

- `213 % 10` ist `3`
- `Collector counting()`

```
public class ZahlenZaehlen {

  private final Zahlen z;
  public ZahlenZaehlen(Zahlen z) {
    this.z = z;
  }

  public Map<Integer,Long> zaehle(int k) {
```

Lösung:

```
    return Stream.generate(z::zufall)
    .limit(k)
    .collect(groupinBy(n -> n % 10, Colectors.counting());
```

```
  }
}
```

Aufgabe 2 [5 Punkte]

(a) [4 Punkte] Schreiben Sie einen Unit-Test für die Methode `zaehle` aus Aufgabe 1. Der Test muss das in der Aufgabenstellung vorgegebene Beispiel verwenden. Markieren Sie in Ihrem Test den kompletten Arrange-Teil.

```
public class ZaehlenTest {
  @Test
  void beispielTest() {
```

Lösung:

```
    // Arrange
    Zahlen z = mock(Zahlen.class);
    when(z.zufall()).thenReturn(21,1,26,11);
    ZahlenZaehlen zz = new ZahlenZaehlen(z);
    // Act
    Map<Integer,Integer> gezaehlt = s.zaehle(4);
    // Assert
    assertThat(summen).contains(entry(1,3), entry(6,1));
```

```
  }
}
```

(b) [1 Punkt] Gelegentlich kritisieren Studierende bei zusammenhängenden Aufgaben in Klausuren, dass sie eine Aufgabe nur lösen können, wenn sie die vorherige Aufgabe gelöst haben. Das ist in der Regel nicht der Fall und in diesem konkreten Fall gibt es sogar Softwareentwickler:innen, die sagen würden, dass wir Aufgabe 2a vor Aufgabe 1 hätten stellen sollen. Welche Entwicklungs-Methode wird von diesen Entwickler:innen mit dieser Aussage empfohlen?

Lösung:
TDD, testgetriebene Entwicklung

Aufgabe 3 [4 Punkte]

Wir wollen eine statische Methode `or` schreiben, die aus zwei Prädikaten ein neues Prädikat berechnet, welches die Ver**oder**ung der Prädikate darstellt.

Beispiel zur Verwendung von `or`:

```
Predicate<Integer> even = z -> z % 2 == 0;
Predicate<Integer> positive = z -> z > 0;
Predicate<Integer> positiveOrEven = Junktoren.or(positive, even);
positiveOrEven.test(5);  // => true (positiv)
positiveOrEven.test(-2); // => true (gerade)
positiveOrEven.test(4); // => true (positiv und gerade)
positiveOrEven.test(-3);  // => false (weder positiv noch gerade)
```

Ein- und Ausgabeprädikate müssen alle denselben Typen als Parameter haben, es sind aber **beliebige Typen** erlaubt, nicht zur Zahlen!

Hinweis: Die einzige abstrakte Methode im Interface `Predicate` heißt `test`.

Geben Sie eine Implementierung von `or`, inkl. Methodenkopf, an:

```
public class Junktoren {
```

Lösung:

```
static <T> Predicate<T> or(Predicate<T> p1, Predicate<T> p2) {
    return v -> p1.test(v) || p2.test(v);
}
```

```
}
```

Aufgabe 4 [4 Punkte]

Ein System zur Blutzuckerkontrolle bei Diabetes soll automatische Reports erzeugen. Dazu wurde die folgende Methode geschrieben:

```
public void report(Patient patient,
      List<Messung> blutzuckerMessungen,
      LocalDate datum) {
    printHeader(patient);
    Double durchschnitt = blutzuckerMessungen.stream()
        .filter(bzm -> datum.equals(bzm.datum()))
        .mapToInt(Messung::wert)
        .average()
        .getAsDouble();
    System.out.printf("Durchschnittlicher Zuckerwert: %.2f\n", durchschnitt);
    if (durchschnitt > 200.0) {
        System.out.println("Achtung! Die Normwerte sind überschritten!");
    }
}
```

Die Methode erzeugt einen Report, der zum Beispiel wie folgt aussehen könnte:

```
Tom Hanks, Typ II Diabetes
Durchschnittlicher Zuckerwert: 225,35
Achtung! Die Normwerte sind überschritten!
```

(a) [3 Punkte] Untersuchen Sie die Methode im Hinblick auf Verletzungen der folgenden uns bekannten Regeln und Prinzipien: DIP, DRY, LCHC, LoD, SLAP, SRP.
Geben Sie an, **welche** der Regeln oder Prinzipien durch den Code sicher **verletzt** sind und **woran** die Verletzung jeweils erkannt werden kann. *Auf* nicht *verletzte Regeln müssen Sie* nicht *eingehen.*

Lösung:

- SLAP: Wir haben eine Mischung von unterschiedlichen Abstraktionsebenen, z. B. die printHeader Methode und die Berechnung und direkte Ausgabe der Resultate.
- SRP: Wir mischen z. B. die Verantwortung der Berichterstattung mit der Berechnung

(b) [1 Punkt] Geben Sie für **eine** der von Ihnen gefundenen Verletzungen an, wie diese behoben werden kann. Sie müssen keinen Code schreiben, aber es muss unmissverständlich klar sein, wie das Ergebnis aussehen sollte.
Den Verstoß gegen **welche Regel** behebt Ihre Verbesserung?

Lösung:

eine der gefundenen Verstöße

Wie wird der Verstoß korrigiert?

Lösung:

- SLAP

```
  public void report(Patient patient,
     List<Messung> blutzuckerMessungen,
     LocalDate datum) {
    printHeader(patient);
    Double durchschnitt = berechneDurchschnitt()
    printDurchschnitt(durchschnitt);
    printWarnung(durchschnitt);
  }
```

- SRP

```
  public void report(Patient patient,
     Double durchschnitt,
     boolean zuHoch) {
    printHeader(patient);
    System.out.printf("Durchschnittlicher Zuckerwert: %.2f\n",durchschnitt);
    if (zuHoch) {
        System.out.println("Achtung! Die Normwerte sind überschritten!");
    }
  }
```

Aufgabe 5 [2 Punkte]

Folgende Klasse riecht nach dem Code Smell *Data Clump*.

```
class Bestellung {

  public void bestellung(String vorname, String nachname,
                         String strasse, String hausnummer, String plz,
                         List<Bestellposition> positionen) {
    // Verarbeitung der Bestellung
  }

  // ...
}
```

(a) [1 Punkt] Wie würden Sie den Code Smell beheben? Sie müssen keinen Code schreiben, aber Ihr Vorgehen muss nachvollziehbar sein.

Lösung:
- Klasse einführen, die die Kundendaten kapselt - Klasse einführen, die die Liste kapselt (List ist hier wie ein primitiver Datentyp zu betrachten)

(b) [1 Punkt] Geben Sie nachvollziehbar einen Grund an, weshalb dieser Code wahrscheinlich besser wartbar ist, wenn der Code Smell **Data Clump** behoben ist.

Lösung:
Eine mögliche Antwort: Die Validierung innerhalb der Fachobjekte wird möglich. Beispielsweise kann geprüft werden, dass die Adresse existiert.

Aufgabe 6 [1 Punkt]

Wir wollen ein Servicesystem für Kühlschränke entwerfen. Wir haben uns entschieden, ein Aggregat zu modellieren, dessen Root-Entität durch die folgende Kühlschrank-Klasse gegeben ist.

Ergänzen Sie die `equals`-Methode entsprechend unserer Modellierungs-Idee.

```
public class Kuehlschrank {
 private final String seriennummer;
 private LocalDate kaufdatum = null;
 private Person eigentuemer = null;

 // passender Konstruktor und hashcode-Methode

 public void verkaufen(LocalDate kd, Person p) {
    this.kaufdatum = kd;
    this.eigentuemer = p;
 }

 public boolean isVerkauft() {
    return eigentuemer != null;
 }

 @Override
 public boolean equals(Object obj) {
  if (this == obj) return true;
  if (!(obj instanceof Kuehlschrank)) return false;
  Kuehlschrank that = (Kuehlschrank) obj;
```

Lösung:

```
  // Es wird nur die Seriennummer für den Vergleich verwendet
  return this.seriennummer.equals(that.seriennummer);
```

```
 }
}
```

Aufgabe 7 [2 Punkt]

Hinweis: Für diese Aufgabe müssen Sie weder Assembler-Programmierung, noch das Erstellen von Präsentationen mit LaTeX beherrschen.

Die Folien und Übungsblätter der Vorlesung Rechnerarchitektur werden mit git versioniert. Neulich hat Janine Markus gefragt, warum er in der Datei `blatt12_aufgabe1.asm` die Zeile `mov eax, 0ffffffffh` geschrieben hat, anstatt die üblichere Notation `mov eax, 0xffffffff` zu verwenden.

(a) [1 Punkt] Mit welchem git-Befehl können wir herausfinden, in welchem Commit zuletzt diese Codezeile verändert wurde?

Lösung:

```
git blame pfad/zu/blatt12\_aufgabe1.asm  # Angabe von blame reicht
```

(b) [1 Punkt] Wir haben herausgefunden, dass die Codezeile mit dem Commit `a21d1f7` hinzugefügt wurde. Der Inhalt des Commits ist im Folgenden angegeben; die Commitnachricht steht oben unterhalb der Date-Zeile; hinzugefügte und entfernte Zeilen sind mit einem einzelnen `-` bzw. `+` gekennzeichnet:

```
$ git show a21d1f7
commit a21d1f7247ebddcb95e5b9b94be7745491028ab5
Author: Markus Brenneis <Markus.Brenneis@uni-duesseldorf.de>
Date:   Thu Jun 31 08:43:25 2021 +0200

    Rechtschreibfehler in Folien behoben

diff --git a/folien/einfuehrung-assembler.tex b/folien/einfuehrung-assembler.tex
index 32be202..f9f483e 100644
--- a/folien/einfuehrung-assembler.tex
+++ b/folien/einfuehrung-assembler.tex
@@ -120,7 +120,7 @@
    \item optional
-   \item Identifizierung einer Zeile ohne Kenntniss der Speicheradresse
+   \item Identifizierung einer Zeile ohne Kenntnis der Speicheradresse
    \item Label ersetzt Adresse
@@ -207,7 +207,7 @@
  \end{frame}
- \begin{frame}[fragile]{Typische Addressierungsfehler}
+ \begin{frame}[fragile]{Typische Adressierungsfehler}
  %\handoutnote{Aufgabe in der Vorlesung.}
diff --git a/uebung/blatt12/loesungen/blatt12_aufgabe1.asm b/uebung/blatt12/loesungen/blatt
index da742af..81aa9ca 100644
--- a/uebung/blatt12/loesungen/blatt12_aufgabe1.asm
+++ b/uebung/blatt12/loesungen/blatt12_aufgabe1.asm
@@ -42,6 +42,8 @@ asm_main:
    dump_regs 4
-   mov eax, 0xffffffff
+   mov eax, 0ffffffffh
    ; Beendigung der main Funktion und Rueckkehr zu C
```

Was hätte Markus beim Committen besser machen können, um Janines Frage jetzt vielleicht beantworten zu können?

Lösung:

Assembler-Änderung und Rechtschriebfehler [sic] in separate Commits mit sprechender Commitnachricht zerlegen.