Grundlagen der Programmierung

Norbert Müller
Universität Trier – Fachbereich IV – Informatik

17. Februar 2022

Bearbeitungszeit: **120 Minuten**

Name _____

Matr.Nr. _____

Rechner _____

- Hiermit bestätige ich, dass ich entsprechend der Prüfungsordnung des von mir belegten Studienganges zur Teilnahme an dieser Klausur zur Vorlesung Grundlagen der Programmierung (Programmierung I) berechtigt bin. Sollte diese Berechtigung nicht vorliegen, bin ich damit einverstanden, dass meine Teilnahme an der Klausur nicht gewertet wird.
- Ich fühle mich gesundheitlich in der Lage, die Klausur anzutreten.

Ort, Datum: _____ Unterschrift: _____

Beachten Sie folgende Regelungen:

- Als Hilfsmittel sind nur die zur Verfügung gestellten Editoren (inkl. Java-Dokumentation) sowie die elektronisch zur Verfügung gestellten Unterlagen (Kurs zur Klausur auf der `moodle`-Plattform) erlaubt. Andere Hilfsmittel wie z.B. das Einloggen in andere Moodle-Kurse oder die Verwendung von Smartphones sind nicht erlaubt.
- In der Klausur sind **112 Punkte** erreichbar. Ziel ist, dass Sie davon etwa **100 Punkte** bearbeiten. Die Zahl der Punkte pro Aufgabe entspricht also in etwa der Bearbeitungszeit (in Minuten).
- Alle Lösungen sind über das Moodle-System abzugeben, d.h. dort abzuspeichern.
- Programme können i.d.R. unter Moodle auch getestet und vor-evaluiert werden. Die Testfälle werden bei der Korrektur allerdings durch neue Fälle ersetzt werden! Abgaben, die nicht compilierbar sind, führen zu deutlichen Punktabzügen!

Aufgabe	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Gesamt*
Möglich	*7*	*6*	*14*	*8*	*8*	*12*	*15*	*15*	*15*	*12*	*112*
Erreicht											

Note: _____

1. Aufgabe: Primitive Datentypen und Termauswertung (7 P)

Bei folgenden Deklarationen von Variablen

```
double eins = 1.0d; double zwei = 2.0d; int drei = 3; int vier = 4;
```

soll der folgende Ausdruck ausgewertet werden:

```
eins / zwei + drei / vier * zwei
//   (1)     (2)     (3)     (4)
```

Geben Sie zunächst an, wie dieser Ausdruck ausgewertet wird, indem Sie im Ausdruck Klammern (entsprechend den Auswertungsregeln in Java) hinzufügen.

Geben Sie zudem an, welchen Typ und welchen Wert die Zwischenresultate der Operationen (1) bis (4) (entsprechend den Auswertungsregeln in Java) an den jeweiligen Positionen haben. Verwenden Sie für die Angabe des Wertes immer Literale des jeweiligen Typs.

Die Abgabe der Antwort erfolgt bei der entsprechenden Aufgabe in Moodle.

2. Aufgabe: Deklarationen und Casts (6 P)

Betrachten Sie die im folgenden angegebenen kurzen Abschnitte aus (evtl. nicht kompilierbaren) Java-Quelltexten. Geben Sie bei den syntaktisch korrekten Textabschnitten den jeweils zugewiesenen Wert an. Bei syntaktisch nicht korrekten Texten geben Sie bitte an, warum sie nicht korrekt sind.

1. `double real = 0,101;`
2. `boolean test = !(0 == 1);`
3. `int value = 010-1;`
4. `string text = "0101";`
5. `int data = 0x10-1;`
6. `int veryLarge = 1234567890;`

Tragen Sie Ihre Antworten bei der entsprechenden Aufgabe in Moodle ein.

3. Aufgabe: Quelltext verstehen und umformulieren

Betrachten Sie die folgende Methode:

```
public static boolean f( int[] data ) {

  if ( data == null ) return false;

  int i = 0;
  int value = data[i];

  while ( i < data.length ) {
      if ( data[i] != value ) return false;
      i++;
  }
  return true;
```

(a) Erklären Sie in Worten das Verhalten dieser Methode (d.h. bei welchen Parametern sie welchen Rückgabewert hat): Wann liefert sie Rückgabewert `true`, wann `false`? Beschreiben Sie dabei NICHT die Arbeit einzelner Zeilen des Quelltextes; zwei oder drei ordentlich formulierte Sätze sollten reichen! Achten Sie auf Sonderfälle! (8 P)

(b) Implementieren Sie den Algorithmus aus `f` analog neu in einer Methode `g`: (6 P)

- Die neue Methode `g` muss für alle(!) möglichen Parameter die gleichen Resultate wie `f` liefern.
- Dabei darf `g` jedoch *keine* `while`-Anweisung enthalten.

Den Quelltext gibt es in Moodle sowohl unter `Aufgabe 03a` als auch unter `Aufgabe 03b`. Ihre Lösung zu Aufgabenteil (a) soll als Freitext in `Aufgabe 03a` eingetragen werden, die Lösung zu Teil (b) soll unter `A3b.java` bei `Aufgabe 03b` in Moodle gespeichert werden; eine zusätzliche Klasse `Evaluation.java` zum Test einfacher Beispiele (aber nicht aller interessanten Fälle!) ist dort vorgesehen.

4. Aufgabe: Strings (8 P)

Schreiben Sie eine Funktion `stringTest`, die zwei formale Parameter `String s1` und `String s2` hat und testet, ob `s1` die Verkettung `t + s2 + t` des Strings `s2` mit einem zweifach zu verwendenden anderen (von Ihnen zu findenden und zunächst unbekannten) String `t` ist. Wenn dies der Fall ist, soll `t` zurückgegeben werden, ansonsten die `null`-Referenz.

Beispiele:

`stringTest("xyzABCxyz", "ABC")` ⇝ `"xyz"`
`stringTest("11222211", "2222")` ⇝ `"11"`
`stringTest("ABC", "ABC")` ⇝ `""`
`stringTest("XaBab", "B")` ⇝ `null`

Sie können davon ausgehen, dass weder `s1` noch `s2` den Wert `null` hat.

Einen Rahmen zur Lösung finden Sie in Moodle unter `Aufgabe 04` in der Datei `A4.java`, mit der Möglichkeit zur testweisen Evaluierung durch `Evaluation.java`. In der dortigen Datei `A4.java` ist die Lösung abzuspeichern.

5. Aufgabe: Rekursion (8 P)

Implementieren Sie die folgende Funktion g als rekursive Funktion mit dem Rückgabetyp `String`, einem Parameter `s` vom Typ `String` und zwei Parametern `m`, `n` vom Typ `int`:

$$g(s,m,n) = \begin{cases} \texttt{null}, & \text{falls } m \geq n \text{ oder } m<0 \text{ ist oder } n>s.length() \\ s.substring[m,n], & \text{falls ansonsten } m+1=n \text{ ist,} \\ g(s,(m+n)/2,n) + g(s,m,(m+n)/2), & \text{in allen anderen Fällen} \end{cases}$$

Beispiele:
```
g( "abcde", 0, 5) = "edcba",   g( "abcde", 0, 3) = "cba",
g( "abcde", 1, 4) = "dcb",     g( "abcde", 2, 3) = "c"
```

(Semantik von g: Es wird aus `s` der Teilstring von Position `m` bis Position `n-1` bestimmt, aber in umgekehrter Zeichenreihenfolge. Dies ist aber für die Lösung nicht relevant.)

Ihre Lösung darf dabei außer den Parametern und evtl. lokalen Variablen keine weiteren Variablen verwenden, insbesondere sind auch Schleifen nicht erlaubt. (Ansonsten wird die Abgabe mit 0 Punkten bewertet, selbst wenn sie die richtigen Resultate liefert. Dies gilt analog für Lösungen, die nur die obigen Beispielresultate reproduzieren.)

Sie dürfen dabei davon ausgehen, dass `s` nicht die `null`-Referenz ist. Einen Rahmen zur Lösung finden Sie in Moodle unter `Aufgabe 05` in der Datei `A5.java`, mit der Möglichkeit zur testweisen Evaluierung durch `Evaluation.java`. In der dortigen Datei `A5.java` ist die Lösung abzuspeichern.

6. Aufgabe: Arrays (12 P)

Implementieren Sie eine Methode `int[] mirrorArray(int[] data)`, die ein (neues!) Array zurückgibt, in dem die Werte aus `data` in umgekehrter Reihenfolge enthalten sind.

Die Inhalte des Feldes `data` müssen dabei erhalten bleiben. Es ist möglich, dass `data` beim Aufruf die `null`-Referenz ist; in diesem Sonderfall soll auch die Null-Referenz zurückgegeben werden.

Beispiel:

Parameterfeld `data`: `{ 1, 2, 3, 4, 5 }`
`mirrorArray(data)`: `{ 5, 4, 3, 2, 1 }`

Parameterfeld `data`: `{ 12345, 54321 }`
`mirrorArray(data)`: `{ 54321, 12345 }`

Parameterfeld `data`: `null`
`mirrorArray(data)`: `null`

(Einen Rahmen zur Lösung finden Sie in Moodle unter `Aufgabe 06`, inklusive einer passenden `main`-Methode und der Möglichkeit zur testweisen Evaluierung. Dort ist auch die Lösung abzuspeichern.)

7. Aufgabe: Collections (15 P)

Vorgegeben ist eine Klasse `ExamResult`, bei der ein Objekt den Namen einer Person (`String name`) und die Note (`int mark`) speichern kann, die die Person bei einer Prüfung erreicht hat.

Ein passender Konstruktor und alle benötigten Getter-Methoden sind hier vorgegeben, so dass diese Klasse ebenso wie die testende Klasse `Evaluation` unverändert bleiben können (und müssen).

Sie sollen nun als Ergänzung dazu eine Klasse `ClassResults` implementieren. Realisieren Sie dafür die folgenden Teilaufgaben:

1. Jedes Objekt der Klasse `ClassResults` soll folgende Informationen speichern: Den Namen einer Prüfung (`String exam`) und zudem eine Liste an Einzelergebnissen (`ArrayList<ExamResult> results`), die die bei dieser Prüfung erreichten Resultate enthält.
2. Implementieren Sie den von `Evaluation` benutzten Konstruktor und ebenfalls die Getter-Methoden `getExam()` und `getResults()`.
3. Eine Methode `void registerMark(ExamResult result)` wird benötigt, um neue Prüfungsergebnisse zur Ergebnisliste eines `ClassResults`-Objektes hinzuzufügen.
4. Außerdem soll jede `ClassResults`-Instanz dazu in der Lage sein, zu einer gegebenen Note alle Personen zu finden, die diese Note erlangt haben. Sie sollen eine ensprechende Methode

   ```
   ArrayList<String> findExaminees(int mark)
   ```

   implementieren, die eine Liste mit den entsprechenden Personen zurückgibt.
5. Zugriff auf die Instanzvariablen der Klasse soll NUR mit dem Konstruktor und den obigen Methoden möglich sein. Diese Methoden und der Konstruktor sollen hingegen frei zugänglich sein.

Zum Testen Ihrer Implementierung ist die Klasse `Evaluation` gegeben. Diese überprüft aber bewußt nur rudimentär, ob die einzelnen Methoden vorhanden sind und bei den wenigen Testfällen die richtigen Rückgabewerte geliefert werden.

8. Aufgabe: Vererbung (15

Schreiben Sie eine konkrete Klasse `Cover` und ein Interface `Stream` wie im folgenden spezifiziert:

Vorgegeben ist eine abstrakte Klasse `Song`. Sie liefert über vier abstrakte Methoden Daten über ein Musikstück: den Titel (`String title()`), die Künstler ((`String artist()`) und die Spieldauer (`double length()`) des Stückes. Außerdem gibt es Zusatzinformationen (`String additionalInfo()`) zum Stück. Diese Klasse und die Testklasse `Evaluation` dürfen nicht verändert werden.

- Implementieren Sie nun eine konkrete Subklasse `Cover` von `Song`. Es geht hierbei um Musikstücke, die bereits früher einmal veröffentlich wurden und jetzt neu gecovert worden sind.
- Zusätzlich zu den Daten über Titel, aktuelle Künstler und Spieldauer, die über `Song` bereits abfragbar sind, soll `Cover` daher folgende Informationen bereitstellen:
  - welche Künstler das Original des Songs eingespielt haben (zu speichern in `String originalArtist`) und
  - in welchem Jahr das geschehen ist (zu speichern in `int originalPublication`).
- Folgende Methoden müssen Sie dazu implementieren:
  - Ein Konstruktor der Form

    ```
    Cover(String title, String artist, double length,
          String originalArtist, int originalPublication)
    ```

  - Zwei getter-Methoden für die zusätzlichen Objektvariablen (also `getOriginalArtist()` und `getOriginalPublication()`)
  - Die vier konkretisierten Methoden für `Song`. Bei `additionalInfo()` soll die Form `"originally by ... from year ..."` mit den zusätzlichen Daten aus `originalArtist` und `originalPublication` gewählt werden; ansonsten sollen die entprechenden Komponenten aus dem Konstruktor unverändert zurückgeben werden.

  Vergeben Sie dabei den Zugriffsmodifikatior `private` und das Attribut `@Override` so oft wie möglich.
- Für eine Austrahlung im Radio wird als Kerninformation nur die Spieldauer benötigt. Daher implementiert `Song` ein (von Ihnen zu schreibendes) Interface `Stream`, das aber nur `length()` bereitstellen soll.

Eine rudimentäre Testklasse `Evaluation` ist vorgegeben, um Ihre Implementierung zu testen. Diese Klasse enthält einige Tests zur Überprüfung, ob die genannten Methoden und Funktionalitäten implementiert wurden.

9. Aufgabe: Exceptions und Strings (15 P)

Implementieren Sie eine Methode

```
public static String checkComparison(String calculation)
```

die überprüft, ob der Parameter `calculation` eine korrekte (und korrekt formatierte) Ungleichung entsprechend folgender Form darstellt:

```
"1 < 2"        "54321 > 12345"        "-1111 < +1111"
```

Im Parameter dürfen dabei nur folgende Teile vorkommen:

- ganze Zahlen in der üblichen Dezimalform, evtl. mit Vorzeichen, so dass `int` als Zahltyp reicht und `Integer.parseInt()` verwendet werden kann (auch zum Test auf Korrektheit der Syntax),
- eines der Zeichen `'>'` und `'<'` (als Vergleichsoperatoren)

Diese drei Teile sind durch ein oder mehrere Leerzeichen voneinander getrennt, so dass problemlos ein `StringTokenizer` zur Zerlegung benutzt werden kann.

Ihre Lösung soll nun folgendes leisten:

- Sollte der String korrekt formatiert sein und zudem die Ungleichung stimmen, so geben Sie folgenden String zurück:

  ```
  valid
  ```

- Sollte der String zwar korrektes Format haben, aber die Werte nicht übereinstimmen, geben Sie einen String im folgenden Format zurück, wobei `left` und `right` die gefundenen Werte sein sollen:

  ```
  left is not smaller than right
  ```

  bzw.

  ```
  left is not greater than right
  ```

- Wenn der String falsch formatiert ist, sollen Sie eine `RuntimeException` werfen, die folgende Nachricht enhhält:

  ```
  syntax error
  ```

Zur Vereinfachung: Auf Zahlüberläufe brauchen Sie nicht zu achten. Der Parameter ist nicht `null`. Eventuelle intern auftretende Exceptions, z.B. aus `Integer.parseInt()`, müssen Sie aber geeignet abfangen.

Einige Beispiele für falsche Gleichungen sind:

```
"-1 > +1"        "54321 < 12345"        "1111 < 1111"
```

Einige Beispiele für falsch formatierte Strings sind:

```
"5<10"           "10 = 5"           "5 < 1 0"
"+ 10 > 9"       "10.0 < 11"        "-5 > - 4"
```

(12 P

10. Aufgabe: Listen

Betrachten Sie die gegebenen Klassen `Liste` und `Elem`, die den in der Vorlesung vorgestellten Varianten sehr ähnlich sind. Sie implementieren eine Minimalfassung einer verketteten Liste von `String`-Werten. Die bereits vorgegebene Methode `addFront(String s)` fügt den String `s` am Anfang der Liste ein.

Erweitern Sie die Klasse `Liste` um die folgenden zwei Methoden:

- Die Methode `ausgeben()` soll einen String mit den Werten sämtlicher Elemente in der Reihenfolge ausgeben, in welcher sie ausgehend von `start` in der Liste enthalten sind. Sie sollen dabei durch einzelne `'+'`-Zeichen getrennt werden. Insbesondere soll nach dem letzten Element kein `'+'` stehen.

  Der erste Aufruf von `ausgeben()` in der `Main.main()`-Methode soll also die folgende Ausgabe produzieren::

  ```
  was+wie+wer+das+die+der
  ```

- Die Methode `int ersetze(String s, String t)` soll in der Liste den String `s` überall durch den String `t` ersetzen und zudem als Rückgabewert angeben, wie oft `s` ersetzt wurde.

  Die weiteren Aufrufe von `ausgeben()` in der `Main.main()`-Methode sollen daher die folgende Ausgabe produzieren:

  ```
  was+wie+wer+das+die+wer
  was+wie+wie+das+die+wie
  was+wie+wie+das+die+wie
  ```

Einen Rahmen zur Lösung finden Sie in Moodle unter `Aufgabe 10`, dort ist auch die Lösung abzuspeichern. Zu dieser Aufgabe finden Sie neben `Liste` und `Elem` auch eine Testklasse `Main` mit einer `main`-Methode in Moodle. Die Klassen `Elem` und `Main` dürfen nicht verändert werden. Sie dürfen zur Vereinfachung davon ausgehen, dass keine `null`-Strings zu verarbeiten sind.