Grundlagen der Programmierung

Norbert Müller
Universität Trier – Fachbereich IV – Informatik

20. Februar 2024

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*	*12*	*15*	*112*
Erreicht											

Note: _____

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

Bei folgenden Deklarationen von Variablen

```
float eins = 1.0f; int zwei = 2; long drei = 3L; double vier = 4.0d;
```

soll der folgende Ausdruck ausgewertet werden:

```
drei / zwei - eins / zwei * drei + vier
    (a)    (b)    (c)    (d)    (e)
```

Geben Sie zunächst an, wie dieser Ausdruck (entsprechend der Auswertungsregeln in Java) ausgewertet wird, indem Sie im Ausdruck 5 öffnende und 5 schließende Klammern hinzufügen.

Beispiel: `eins + zwei + drei` würde ausgewertet als `((eins + zwei) + drei)`

Geben Sie zudem an, welchen Typ und welchen Wert die Zwischenresultate der Operationen (a) bis (e) **bei der von Ihnen gewählten Klammerung** 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. `boolean bit = (12 != 34);`
2. `int value = 1.234;`
3. `double val = (int) 123.4;`
4. `int byte = 123;`
5. `string text = "1234";`
6. `int array[] = {12,34};`

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

3. Aufgabe: Quelltext verstehen und umformulieren

Betrachten Sie die folgende Methode:

```
public static int f( int[] data, int value ) {

  if ( data == null ) throw new IllegalArgumentException();

  int i = data.length - 1;

  while (i != 0 ) {
      i--;
      if ( data[i] == value ) return 0;
  }
  return 1;
}
```

(a) Erklären Sie in Worten, bei welchen Kombinationen der Parameter `data` und `value` diese Methode den Wert `1` zurückgibt, wann den Wert `0` und welche anderen Programmverläufe es gibt. Drei oder vier ordentlich formulierte Sätze sollten dazu reichen. Achten Sie auf Sonderfälle! (8 P)

Beschreiben Sie dabei *nicht* den Ablauf des Algorithmus, also nicht die einzelnen Zeilen des Quelltextes! Sie sollen stattdessen nur das beim Aufruf sichtbare Verhalten beschreiben.

(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 einen formalen Parameter `String s` hat und testet, ob `s` die Verkettung `t + t` eines anderen (zunächst unbekannten und von Ihnen zu bestimmenden) Strings `t` ist. Wenn dies der Fall ist, soll `t` zurückgegeben werden, ansonsten die `null`-Referenz.

Beispiele:
`stringTest("xyzxyz")` ⇝ `"xyz"`
`stringTest("1111111111")` ⇝ `"11111"`
`stringTest("abababab")` ⇝ `null`

Achten Sie darauf, dass der Parameter `s` hier sogar `null` sein kann!

(Einen Rahmen zur Lösung finden Sie in Moodle unter `Aufgabe 04`, dort ist auch die Lösung in der Datei `A4.java` abzuspeichern. Eine passende `main`-Methode mit der Möglichkeit zur testweisen Evaluierung ist in einer Klasse `Evaluation` bereits vorgegeben.)

5. Aufgabe: Rekursion (8 P)

Implementieren Sie die folgende Funktion g als rekursive Funktion mit dem Rückgabetyp `String` und einem Parameter `s` vom Typ `String`:

$$\texttt{g(s)} = \begin{cases} \texttt{s}, & \text{falls die Länge von } \texttt{s} \text{ eine gerade Zahl ist,} \\ \texttt{g(s1) + "\#" + g(s2)}, & \text{in allen anderen Fällen, wobei } \texttt{s1} \text{ und } \texttt{s2} \text{ sich ergeben aus } \texttt{s1.length() = s2.length() = (s.length()-1)/2} \\ & \text{und } \texttt{s = s1+c+s2} \text{ mit einem einzelnen(!) Character } \texttt{c}. \end{cases}$$

`c` ist also dabei der Character genau in der Mitte von `s`, der durch `"#"` ersetzt wird; der vordere und hintere Teil von s werden rekursiv analog behandelt.

Beispiele:

```
   g("ab")  ⇝ "ab",            g("abc")  ⇝ "###",
g("abcde")  ⇝ "ab#de", g("abcdefghijk")  ⇝ "ab#de#gh#jk"
```

Ihre Lösung darf dabei außer den Parametern und evtl. lokalen Variablen keine weiteren Variablen verwenden, auch Schleifen sind 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 den Wert `null` hat. 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: Exception und lokale Klassen (15 P)

Vorgegeben ist eine Klasse `Evaluation` mit zwei Methoden: Eine kurze `main`-Methode und eine Methode `void exceptionTrigger()`.

- `main` ruft eine Methode `exceptionCheck` auf, die Sie noch schreiben sollen.
- `void exceptionTrigger()` liest eine `int`-Zahl n von der Konsole ein. Sollte die eingegebene Zahl nicht zwischen 1 und 5 liegen, läuft `exceptionTrigger` ohne Probleme durch. Liegt n aber in diesem Bereich, wird in Abhängigkeit von n eine von fünf unterschiedlichen Exceptions entstehen.

Schreiben Sie nun in der Klasse `Checker` die Methode `int exceptionCheck()` und zudem eine lokale Exception-Klasse `NoProblemException` mit folgenden Eigenschaften:

- `exceptionCheck` soll `exceptionTrigger` aufrufen und durch geeignetes Fangen der jeweils geworfenen Exception die eingelesene Zahl bestimmen und an `main` zurückgeben.
- Welche Zahl bei `exceptionTrigger` zu welcher Exception führt, können Sie im Quelltext der Klasse `Evaluation` und der Evaluation unter Moodle leicht erkennen.
- Wenn `exceptionTrigger` also zum Beispiel eine 1 einliest, wird eine `ArithmeticException` geworfen. Dann muss `exceptionCheck` auch eine 1 zurückgeben.
- Ist die von `exceptionTrigger` gelesene Eingabe nicht zwischen 1 und 5, so läuft die Methode *ohne* Exception ab. In solchen Fällen soll nun von `exceptionCheck` selbst eine `NoProblemException` (als Subklasse von `RuntimeException`) geworfen werden.

An der Klasse `Evaluation` dürfen Sie keine Änderungen vornehmen; Ihre Lösung ist komplett in der Datei `Checker.java` abzuspeichern.

8. Aufgabe: Vererbung (15 P)

Vorgegeben ist eine Klasse `Song`. Sie speichert über ein Musikstück Informationen wie den Titel (`String title`) und die Musikgruppe (`String band`). Folgende Methoden werden dabei bereitgestellt:

- Ein Konstruktor der Form

  ```
  Song(String title, String band)
  ```

- Die getter-Methoden für einen `Song` (also `getTitle()` und `getBand()`)

Implementieren Sie passend dazu ein Interface `Playlist`. Der Zweck dieses Interfaces ist die Speicherung einer Sammlung von Musikstücken (`Song`). Daher soll das Interface die folgenden drei Methoden vorsehen:

- `void addSong(Song mySong)`
- `Song play(String title)`
- `int getSize()`

Leiten Sie schließlich die Klasse `SimplePlaylist` vom Interface ab. Die Klasse `SimplePlaylist` speichert Musikstücke in einem Array (`Song[] songs`). Dabei wird die Größe des Arrays im Konstruktor angegeben. Zudem speichert sie die Anzahl der über `addSong` hinzugefügten Musikstücke in einer geeigneten Variablen `size`.

Implementieren Sie `SimplePlaylist` wie folgt:

- Implementieren Sie einen Konstruktor `public SimplePlaylist(int maxsize)`, der als Parameter die Größe des Arrays hat.
- Implementieren Sie in `SimplePlaylist` die Methode `addSong`, die ihren Parameter-Song zur Playlist-Instanz hinzufügt. Sie können dabei eine beliebige Reihenfolge der Speicherung im Array `songs` wählen, ein Anfügen bei der ersten "freien" Stelle ist sicherlich am einfachsten. Wenn kein freier Platz mehr im Array existiert, geben Sie den folgenden String auf der Konsole aus und ignorieren ansonsten den Aufruf:

  ```
  Playlist full
  ```

- `getSize()` kann als getter-Methode implementiert werden.
- Zum Schluss sollen Sie in `SimplePlaylist` die Methode `play` des Interfaces implementieren. Diese Methode soll im Array `songs` nach einem Stück mit diesem Titel suchen und dieses, wenn gefunden, zurückgeben. Wenn der gesuchte Titel nicht in `songs` enthalten ist, so geben Sie die `null`-Referenz zurück.

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: Objekte (12 P)

Betrachten Sie die Klassen `Client` und `Parser`. Die Klasse `Client` speichert für einen Kunden eine ID, den Namen und eine Geldsumme, die vom Kunden für Waren ausgegeben wurde. Außerdem sind die Getter-Funktionen und eine `toString()`-Funktion bereits vorgegeben.

Erweitern Sie die Klasse `Parser` um eine Methode `createClient(String string)`, die in der Lage ist, Strings der folgenden Form zu verarbeiten und als Objekte zu speichern:

```
{cnr:1234;name:Nyso;value:200}
{cnr:1234;value:200;name:Nyso}
{cnr:1234   ;value   :200    ;name   : Nyso}
```

Berücksichtigen Sie, dass die Reihenfolge der Elemente des Triples unterschiedlich sein kann und ebenfalls unnötige Leerzeichen in den Strings vorkommen können.

Weitere Fehler müssen Sie nicht betrachten, da nur die erwähnten Fehler vorkommen!

10. Aufgabe: Stacks (15 P)

Betrachten Sie die gegebenen Klassen `Stack` und `Elem`, die den in der Vorlesung vorgestellten Varianten sehr ähnlich sind. Sie enthalten einen Teil einer Stack-Implementierung aus verketteten Listenelementen, wobei jedoch der Typ `Integer` für die Datenkomponente fest voreingestellt ist. Eine `push`-Methode ist bereits vorgegeben, ebenso wie eine Methode `toString()`.

Erweitern Sie die Klasse `Stack` um die folgenden drei Methoden `pop()`, `invert()` und `removeZeroes()`:

1. Die Methode `pop()` ergänzt das Verhalten der Klasse um die (noch fehlende) Pop-Operation, so dass tatsächlich eine lauffähige Stack-Inplementierung entsteht.

   Bei einem leeren Stack soll `pop()` der Einfachheit halber den Zahlwert -123456 zurückgeben.

   In der `Evaluation.main()`-Methode wird dies beim Testfall 1 (dh. Eingabe von 1) grob getestet. Es sollte dabei die folgende Ausgabe produziert werden:

   ```
   Startwerte: [ 15 10 5 ]
   Resultat: 15 10 5 -123456
   ```

2. Die Methode `invert()` soll das Vorzeichen aller im Stack enthaltenen Einträge umdrehen, d.h. aus positiven Zahlen werden negative Zahlen und umgekehrt. Die Reihenfolge der Werte soll ansonsten unverändert bleiben.

   In der `Evaluation.main()`-Methode wird dies beim Testfall 2 grob getestet. Es sollte dabei die folgende Ausgabe produziert werden:

   ```
   Startwerte: [ -4 -3 0 1 2 0 ]
   Resultat: [ 4 3 0 -1 -2 0 ]
   ```

3. Die Methode `removeZeroes()` soll im Stack alle Einträge entfernen, die den Wert 0 haben. Alle anderen Werte sollen in der Original-Reihenfolge erhalten bleiben.

   In der `Evaluation.main()`-Methode wird dies beim Testfall 3 grob getestet. Es sollte dabei die folgende Ausgabe produziert werden:

   ```
   Startwerte: [ 0 0 1 0 2 0 ]
   Resultat: [ 1 2 ]
   ```

   Beim Testfall 4 werden wichtige Sonderfälle von `removeZeroes()` getestet. Es sollte dabei die folgende Ausgabe produziert werden:

   ```
   Startwerte: [ 0 0 0 ]
   Resultat: [ ]
   ```

Einen Rahmen zur Lösung finden Sie in Moodle unter `Aufgabe 10`, dort ist auch die Lösung abzuspeichern. Zu dieser Aufgabe finden Sie neben `Stack` und `Elem` auch eine Testklasse `Evaluation` mit einer `main`-Methode in Moodle. Die Klassen `Elem` und `Evaluation` dürfen nicht verändert werden.