Test-Klausur zur Veranstaltung

Objektorientierte Programmierung

im Sommersemester 2024

universität ulm

22. Mai 2024, 12:30 Uhr

Bearbeitungszeit: 30 min

Florian Sihler, Raphael Straub,
Prof. Matthias Tichy

Institut für Softwaretechnik und Programmiersprachen

Musterlösung

Aufgabe 1 — Wissensfragen

2 + 2 + 2 = 6 Punkte

Kreuzen Sie zu jeder Frage die korrekte(n) Antwortmöglichkeit(en) an. Gehen Sie bei gegebenen Code-Schnipsel davon aus, dass diese jeweils korrekt in umliegenden Code eingebettet sind, also beispielsweise in der `main`-Methode einer Klasse stehen.

a) Gegeben sei der folgende Code. Es gibt nur *eine* korrekte Antwort: 2 Punkte

```
1 var a = 35;
```

- ◯ Der Typ der Variable a ist `char`.
- ◯ Der Typ der Variable a ist `long`.
- ◯ Der Typ der Variable a ist `byte`.
- ◯ Der Typ der Variable a ist `String`.
- ☒ Der Typ der Variable a ist `int`.
- ◯ In Java gibt es *keine* Typinferenz.

b) Betrachten Sie erneut folgenden Code, es gibt nur *eine* korrekte Antwort: 2 Punkte

```
1 byte[] i = {1, 2, 3};
2 do {
3    System.out.print(i[i.length - 1]);
4    i[i.length - 1] = 0;
5 } while (i.length < 3);
```

- ◯ Der Code erzeugt die Ausgabe „123“.
- ☒ Der Code erzeugt die Ausgabe „3“.
- ◯ Der Code erzeugt die Ausgabe „321“.
- ◯ Der Code erzeugt die Ausgabe „1“.
- ◯ Der Code erzeugt die Ausgabe „300“.
- ◯ Der Code erzeugt einen Kompilierfehler.

c) Kreuzen Sie alle Aussagen an, die für Java korrekt sind (mehrere Antworten möglich): 2 Punkte

- ☐ Variablen sind in Java standardmäßig unveränderlich.
- ☒ Arrays werden in Java auf dem Heap abgelegt.
- ☒ Java hat einen Garbage Collector.
- ☐ Das Liskovsche Substitutionsprinzip findet in Java *keine* Anwendung.
- ☐ Das Typsystem von Java macht Tests überflüssig.
- ☒ Alle Probleme die wir mit Schleifen lösen können, können wir auch mit Rekursion lösen.
- ☐ Abstrakte Klassen können in Java mit dem Schlüsselwort `new` instanziiert werden.

Aufgabe 2 — Imperative Programmierung **5 + 5 = 10 Punkte**

Sie dürfen für die folgenden Aufgaben Hilfsfunktionen implementieren!

a) Implementieren Sie eine Funktion `int getMaximum(int[])`, die das Maximum eines Arrays von ganzen Zahlen zurückgibt. Sie dürfen davon ausgehen, dass das Array mindestens ein Element enthält. 5 Punkte

Beispiele:
- `getMaximum(new int[] {1, 2, 3, 4, 5})` gibt `5` zurück.
- `getMaximum(new int[] {3, 2, 42, 1, -3})` gibt `42` zurück.

Lösung:

```java
public static int getMaximum(int[] arr) {
    int max = arr[0];
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

Auslassen der Qualifier (`public static`) der Methode führt hier zu keinem Abzug. Mögliche Alternativen (diese sind zwar (noch) nicht Inhalt der Vorlesung, aber in der Klausur gestattet):

- `Arrays.stream(arr).max().getAsInt();`
- `Arrays.sort(arr); return arr[arr.length - 1];`
  Hinweis: Diese Lösung ist nicht optimal, da sie das Array verändert, dies wurde von der Aufgabe allerdings nicht verboten.

Allgemein akzeptieren wir aber jede korrekte Lösung. Ein `null`-check ist demnach nicht notwendig.

b) Schreiben Sie eine Funktion `printDiamond(int n)`, die ein Diamantmuster der Größe `n` auf der Konsole ausgibt. Das Muster soll aus Sternen („*“) bestehen. Sie dürfen davon ausgehen, dass `n` immer ≥ 1 ist. 5 Punkte

Beispiel:

- `printDiamond(3)` soll folgendes Muster ausgeben (Leerfelder werden hier der Übersichtlichkeit wegen mit einem Punkt „.“ dargestellt):

```
...*
..***
.*****
..***
...*
```

Lösung:

Das Beispiel fordert ein anfängliches Leerfeld, dies lässt sich aber leicht mit einem `System.out.print(" ")` oder einer Anpassung der repeat-Wiederholungen lösen.

Der naive Ansatz wie schon auf dem Übungsblatt (hier allerdings mit `String::repeat`):

```
1  public static void printDiamond(int n) {
2      for (int i = 1; i <= n; i++) {
3          System.out.print(" ");
4          System.out.println(" ".repeat(n - i) + "*".repeat(2 * i - 1));
5      }
6      for (int i = n - 1; i > 0; i--) {
7          System.out.print(" ");
8          System.out.println(" ".repeat(n - i) + "*".repeat(2 * i - 1));
9      }
10 }
```

Wie schon zuvor, gibt es für die keinen Abzug für das Auslassen der Qualifier (`public static`). Wie schon zuvor, akzeptieren wir aber jede korrekte Lösung.

Man kann alternativ auch eine mathematische Funktion schreiben und die so entstehende Diamanten-Fläche füllen.

```
1  public static void printDiamond(int n) {
2      for(int y = -n + 1; y < n; y++) {
3          System.out.print(" ");
4          for(int x = -n + 1; x < n; x++) {
5              if(Math.abs(x) + Math.abs(y) < n) {
6                  System.out.print("*");
7              } else {
8                  System.out.print(" ");
9              }
10         }
11         System.out.println();
12     }
13 }
```

Aufgabe 3 — Fehler finden **5 Punkte**

Markieren Sie in folgendem Code alle Kompilier- und Laufzeitfehler und verwenden Sie die Zeilen unten um kurz zu beschreiben, was der Fehler ist **und** wie man diesen korrigieren kann.

Lösung:

In der Datei `Errors.java`:

```
1  class Errors {
2     public String foo() { return "foo"; }
3
4     public int double(int x) { return (2*)x; }
5
6     public static void main() {
7        Errors e = new MegaErrors();
8        System.in.print(e.foo());
9     }
10 }
```

In der Datei `MegaErrors.java`:

```
1  class MegaErrors implements Errors {
2     public String foo(int i) {
3        return "foo-" + i;
4     }
5  }
```

Beachten Sie, dass die Anzahl der Zeilen nicht mit der Anzahl der Fehler übereinstimmen muss und Ihnen mehr Platz zur Verfügung stehen kann.

- Der Bezeichner `double` ist nicht erlaubt (Keyword) $\Longrightarrow$ Umbenennung
- Der Ausdruck `(2*)` ist in Java nicht erlaubt $\Longrightarrow$ Entfernen der Klammern, bzw. `(2 * x)`.
- Die Methode `print(String)` existiert nicht auf `System.in` $\Longrightarrow$ `System.out.print`
- Damit `MegaErrors` von `Errors` erben kann, benötigen wir das Keyword `extends`.
-
-

Lösung:

Optional erlauben wir auch: Wenn `Errors` und `Mega-Errors` nicht im gleichen Ordner sind, muss `Errors` durch `public` sichtbar gemacht werden.

Aufgabe 4 — Generische Freuden

2 + 1 + 2 = 5 Punkte

Machen Sie sich zunächst mit dem folgenden Code vertraut:

```java
public class Pair<F, S> {
    public final F fst;
    public final S snd;

    public Pair(F fst, S snd) {
        this.fst = fst;
        this.snd = snd;
    }
}
```

Geben Sie im Folgenden jeweils nur die Ausdrücke an, die notwendig sind um die jeweilige Aufgabe zu lösen. Folgefehler werden Ihnen dabei nicht zur Last gelegt. Die Verwendung von sogenannten „Raw Types“ ist in dieser Aufgabe allerdings nicht erlaubt, geben Sie die *passendsten* Typen vollständig an.

a) Erstellen Sie eine Instanz von `Pair` mit `"Hello"` als erstem und `42` als zweitem Element und speichern Sie diese in einer Variable `p`. **2 Punkte**

```java
Pair<String, Integer> p = new Pair<String, Integer>("Hello", 42);
```

Lösung:

Wir erlauben weder `Pair<String, Byte>` noch andere numerische Typen für `42` (hier findet keine implizite Typkonvertierung statt!). Ebenso gibt es Abzug für `Pair<Object, Object>`, da wir nach dem *passendsten* Typen fragen.

b) Geben Sie den Wert des ersten Elements von `p` aus. **1 Punkt**

```java
System.out.println(p.fst);
```

c) Erzeugen Sie eine Kopie von `p` und speichern Sie diese in einer neuen Variable `p2`. **2 Punkte**

```java
Pair<String, Integer> p2 = new Pair<String, Integer>(p.fst, p.snd);
```