Programmiertechnik 1 – Probeklausur

Bearbeitungszeit 120 Minuten. Als Hilfsmittel ist ein Spickzettel im Format DIN-A4 zugelassen.

Aufgabe 1: Zahlensysteme *(8 Punkte)*

a) Geben Sie die Dezimalzahl **`107`** in folgenden Schreibweisen an: *(4 Punkte)*
- als ganzzahliges binäres Java-Literal
- als ganzzahliges oktales Java-Literal
- als ganzzahliges hexadezimales Java-Literal
- als Java-Gleitkommaliteral

b) Geben Sie die Dezimalzahl **`-107`** als Binärzahl in 16-stelliger Zweierkomplementdarstellung an. *(2 Punkte)*

c) Geben Sie die Dezimalzahl **`0,`**$\overline{\mathbf{3}}$ (Nullkommadrei-Periode, also **`0,3333...`**) als Binärzahl mit Festkomma an. *(2 Punkte)*

Aufgabe 2: Datentypen *(16 Punkte)*

a) Betrachten Sie die folgenden Variablenzuweisungen:

```
a = 1 / 2
b = 3 - 4.5
c = 6 == 7
d = "89" + '0'
```

Geben Sie zu jeder Variablen an:
- den Typ, den sie haben muss, damit ihr Typ genau dem Typ des zugewiesenen Werts entspricht
- den Wert, den sie nach der Zuweisung hat. *(4 Punkte)*

b) Erklären Sie mit einer kleinen Grafik den Unterschied zwischen einer Variablen mit Werttyp und einer Variablen mit Referenztyp. Nennen Sie für beide Arten von Typen je ein Beispiel. *(4 Punkte)*

c) Der allgemeinste Typ in Java ist **`java.lang.Object`**.
Erklären Sie, warum die beiden Zuweisungen in der folgenden Anweisungsfolge korrektes Java sind:

```
Object x;
x = new Beispiel();
x = 1;
```

(4 Punkte)

d) In Java gibt es für alles eine String-Darstellung vom Typ **`java.lang.String`**.
Erklären Sie jeweils an einem Beispiel,
- wie man die String-Darstellung einer ganzen Zahl vom Typ **`int`** erhält
- wie man die String-Darstellung eines beliebigen Objekts erhält.

(4 Punkte)

Aufgabe 3: Ausdrücke *(8 Punkte)*

a) Gegeben ist der folgende Java-Ausdruck: **`a = ++b + c-- >>> 1`**

Zeichnen Sie den Auswertungsbaum des Ausdrucks und nummerieren Sie darin die Knoten entsprechend der Ausführungsreihenfolge (Nummer Eins für den als erstes auszuführenden Operator). *(5 Punkte)*

b) Angenommen, alle drei Variablen im Ausdruck aus a) haben den Typ **`int`** und **`a`** ist mit dem Wert **`10`** initialisiert, **`b`** mit dem Wert **`20`** und **`c`** mit dem Wert **`30`**.

Welche Werte haben die drei Variablen jeweils nach der Auswertung des Ausdrucks? *(3 Punkte)*

Aufgabe 4: Anweisungen *(17 Punkte)*

Betrachten Sie die folgende Main-Klasse:

```
 1  public final class Anweisungen {
 2      private Anweisungen() { }
 3
 4      public static void main(String[] args) {
 5          System.out.println(max());
 6          System.out.println(max(1));
 7          System.out.println(max(2, 3));
 8          System.out.println(max(4, 5, 6));
 9      }
10
11      private static Integer max(Integer n, Integer m) {
12          return n.intValue() > m.intValue() ? n : m;
13      }
14
15      private static Integer max(Integer... zahlen) {
16          Integer m;
17          switch (zahlen.length) {
18          case 0:
19              m = null;
20              break;
21          case 1:
22              m = zahlen[0];
23              break;
24          default:
25              m = zahlen[0];
26              for (Integer n : zahlen) {  // for-each-Schleife
27                  m = max(m, n);
28              }
29              break;
30          }
31
32          return m;
33      }
34  }
```

a) Formulieren Sie die erste **max**-Methode (Zeile 11 bis 13) ohne Bedingungsoperator. *(2 Punkte)*

b) Formulieren Sie die zweite **max**-Methode (Zeile 15 bis 33) mit **if**-Anweisungen statt der **switch**-Anweisung und mit einer **for**-Schleife mit Index-Laufvariable statt der for-each-Schleife. *(8 Punkte)*

c) Formulieren Sie Ihre **for**-Schleife aus b) als **while**-Schleife. *(3 Punkte)*

d) Geben Sie für jede der Zeilen 5, 6, 7 und 8 an, welche der beiden **max**-Methoden dort aufgerufen wird und welchen Wert der Methodenaufruf jeweils liefert. *(2 Punkte)*

e) Wie realisiert der Java-Compiler die Argumentübergabe beim **max**-Aufruf in Zeile 8? *(2 Punkte)*

Aufgabe 5: Klassen *(12 Punkte)*

Betrachten Sie die folgende Main-Klasse **`BeispielTest`**, die eine Utility-Klasse **`Beispiel`** testet:

```
public final class BeispielTest {
    private BeispielTest() { }

    public static void main(String[] args) {
        int i = aufgabe5.Beispiel.methode();
        int j = aufgabe5.Beispiel.methode(i);
        int k = aufgabe5.Beispiel.KONSTANTE;
        aufgabe5.Beispiel.variable = i + j + k;
    }
}
```

a) Wie muss die Datei mit dem Quellcode von **`Beispiel`** heißen und wo muss sie im Dateisystem liegen? *(1 Punkt)*

b) Geben Sie eine übersetzbare und zusammen mit dem Testprogramm funktionstüchtige Implementierung der Utility-Klasse **`Beispiel`** an. *(8 Punkte)*

c) Erklären Sie am Beispiel der Klasse **`Beispiel`** das Überladen von Methoden (Overloading). *(1 Punkt)*

d) Was muss man im Kopf der Quellcodedatei von **`BeispielTest`** ergänzen, damit man

- in **`main`** statt **`aufgabe5.Beispiel`** immer einfach **`Beispiel`** schreiben kann?
- in **`main`** statt **`aufgabe5.Beispiel.KONSTANTE`** einfach **`KONSTANTE`** schreiben kann?

(2 Punkte)

Aufgabe 6: Objekte *(17 Punkte)*

a) Programmieren Sie eine instanziierbare Klasse **`Aufgabe`**, die pro Instanz den Titel einer Klausuraufgabe sowie ein Feld mit der Anzahl der Punkte pro Teilaufgabe kapselt. Die Klasse soll sicherstellen, dass eine Aufgabe insgesamt und jede ihrer Teilaufgaben für sich mindestens einen Punkt erhält. Die Klasse soll nicht als Oberklasse verwendbar sein. Leiten Sie die Methoden der Klasse aus den Aufrufen in der folgenden **`main`**-Methode ab: *(11 Punkte)*

```
1    public static void main(String[] args) {
2        Aufgabe[] klausur = {
3            new Aufgabe("Zahlensysteme", new int[] {4, 2, 2}),
4            new Aufgabe("Datentypen", new int[] {4, 4, 4, 4})
5        };
6        for (int i = 0; i < klausur.length; ++i) {
7            System.out.print("Aufgabe " + (i + 1));
8            int n = klausur[i].getAnzahlTeile();
9            if (n > 1) {
10               System.out.printf(" a) bis %c)", 'a' + n - 1);
11           }
12           System.out.printf(": %s (%d Punkte)%n",
13                             klausur[i].getTitel(),
14                             klausur[i].getGesamtPunkte());
15       }
16   }
```

Aufgaben 1 und 2 dieser Klausur mit den Punkten ihrer drei bzw. vier Teilaufgaben

Konsolenausgabe:

```
Aufgabe 1 a) bis c): Zahlensysteme (8 Punkte)
Aufgabe 2 a) bis d): Datentypen (16 Punkte)
```

b) Erklären Sie am Beispiel Ihrer Klasse **`Aufgabe`**, was **`this`** ist. Wo und wie ist **`this`** definiert, wann und womit wird **`this`** initialisiert und wo kann **`this`** benutzt werden? *(4 Punkte)*

c) Wie kann man den Ausdruck **`"Aufgabe " + (i + 1)`** aus **`main`** Zeile 7 ohne den Operator **`+`** mit der Klasse **`java.lang.StringBuilder`** realisieren? *(2 Punkte)*

Aufgabe 7: Objektorientierung *(6 Punkte)*

a) Nennen Sie die Kennzeichen der objektorientierten Programmierung. *(2 Punkte)*

b) Woran erkennt man, dass ein bestimmtes Kennzeichen der Objektorientierung in einem Java-Programm vorkommt? Geben Sie zur Beantwortung für jedes Kennzeichen aus a) ein kurzes Codebeispiel. *(4 Punkte)*

Aufgabe 8: Schnittstellen *(16 Punkte)*

In der Vorlesung haben wir ein Feld von Strings mit der Bibliotheksmethode **`java.util.Arrays.sort`** sortiert. Allerdings beruhte die Sortierreihenfolge auf der natürlichen Ordnung der Zeichen, also der Reihenfolge der Zeichencodes. Beim Sortieren von deutschen Wörtern möchte man aber typischerweise Groß- und Kleinbuchstaben sowie Umlaute und Nicht-Umlaute gleich behandeln, also a, A, ä und Ä sollen gleich sein usw. Außerdem soll das ß als ss (Doppel-S) behandelt werden.

Das Programm aus der Vorlesung lässt sich mit einem **`Comparator`**-Funktionsobjekt an die in der deutschen Sprache übliche Sortierreihenfolge anpassen:

```
public final class StringSort {
    private StringSort() { }

    public static void main(String[] args) {
        java.util.Arrays.sort(args, new GermanComparator());

        for (String s : args) {
            System.out.println(s);
        }
    }
}
```

Programmieren Sie die Klasse **`GermanComparator`**.

- **`GermanComparator`** muss die Schnittstelle **`java.util.Comparator<String>`** implementieren
- **`GermanComparator`** soll eine private Klassenmethode

  `String normalize(String s)`

  enthalten, die alle Buchstaben aus dem String **`s`** mittels **`Character.toUpperCase(char)`** in Großbuchstaben wandelt und dann im Falle eines Umlauts durch den Nicht-Umlaut (Ä wird A usw.) bzw. im Falle von ß durch SS ersetzt.

  Die Methode liefert den so normalisierten String als Rückgabewert.
- **`GermanComparator`** muss die Methode

  `int compare(String s1, String s2);`

  aus der Schnittstelle **`java.util.Comparator<String>`** überschreiben.

  Normalisieren Sie im Rumpf der Methode zuerst die beiden Strings mit Hilfe von **`normalize`** und führen Sie dann darauf einen einfachen String-Vergleich durch.

  Der Rückgabewert der Methode muss wie bei Vergleichsmethoden üblich kleiner **`0`** sein, wenn **`s1 < s2`**, größer **`0`**, wenn **`s1 > s2`** und **`0`** sonst.