

Vorname

Nachname

Matrikel-Nr

Tragen Sie Ihre Lösungen zu den **Aufgaben 1 bis 3** in das folgende **Formular** ein. Schreiben Sie Ihre Lösungen zu den **Aufgaben 5 und 6** auf je ein extra Blatt und kennzeichnen Sie diese Blätter oben links mit Ihrem **Nachnamen**.

Formular für die **Lösungen** zu den Aufgaben 1 bis 3:

1.1. (Ausgabe für Eingabe –7)	3.1.
1.2. (Ausgabe für Eingabe 9)	3.2.
1.3. (Ausgabe für Eingabe 12)	3.3.
2.1. (Module?)	3.4.
2.2. (int-Variablen?)	3.5.

Aufgabe 1 (20 Punkte): Betrachten Sie das folgende Programm:

```

1 public class Aufgabe1 {
2     // -----
3     public static void main(String[] susi) {
4         int ein = 0;
5         System.out.print("Eine positive gerade Ganzzahl? ");
6         try {
7             ein = liesGerade();
8             System.out.println(ein + " ist in Ordnung!");
9             if (ein == 0) break;
10        }
11        catch (UngeradeException ue) {
12            System.out.println(ue.getZahl() + " ist nicht gerade!");
13        }
14        finally {
15            System.out.println("Geschafft!");
16        }
17    } // main
18    // -----
19    static class UngeradeException extends Exception {
20        private int zahl;
21        public UngeradeException(int n) {zahl = n;}
22        public int getZahl() {return zahl;}
23    }
24    // -----
25    static class NegativException extends Exception {
26        private int zahl;
27        public NegativException(int n) {zahl = n;}
28        public int getZahl() {return zahl;}
29    }
30    // -----
31    public static int liesGerade() throws UngeradeException {
32        int erg = 0;
33        try {
34            erg = liesNat();
35        }
36        catch (NegativException ae) {
37            System.out.println("Falsche Eingabe!");
38            erg = - ae.getZahl();
39        }
40        if (erg % 2 != 0) throw new UngeradeException(erg);
41        return erg;
42    } // liesGerade
43    // -----
44    public static int liesNat() throws NegativException {
45        int erg;
46        try {
47            erg = Lesen.liesInt();
48        }
49        catch (java.io.IOException ioe) {
50            erg = 0;
51        }
52        if (erg < 0) {
53            System.out.println(erg + " ist negativ!");
54            throw new NegativException(erg);
55        }
56        return erg;
57    } // liesNat
58    // -----
59 } // class Aufgabe1

```

- 1.1. Angenommen, der Benutzer startet dieses Programm und gibt auf die entsprechende Aufforderung hin (siehe Zeile 5) die Zahl **-7** ein. Was wird daraufhin zum Bildschirm ausgegeben?
- 1.2. Ebenso, aber der Benutzer gibt **9** ein.
- 1.3. Ebenso, aber der Benutzer gibt **12** ein.

Aufgabe 2 (15 Punkte): Betrachten Sie die folgenden Klassenvereinbarungen:

```
1 class Klasse1 {
2     static int anna = 1;
3     static int berta = 2;
4     int carl = 10;
5 }
6
7 class Klasse2 extends Klasse1 {
8     static int dora = 3;
9     int emil = 20;
10    int felix = 30;
11 }
12
13 public class Test12 {
14     public static void main(String[] s) {
15         Klasse1 ob11 = new Klasse1();
16         Klasse1 ob12 = new Klasse1();
17         Klasse2 ob21 = new Klasse2();
18         Klasse2 ob22;
19         ...
20     } // main
21 } // class Test12
```

Führen Sie das Programm **TestAB** aus ("im Kopf" oder "mit Papier und Bleistift") bis Sie die **Zeile 19** erreichen. Beantworten Sie für diesen Moment die folgenden Fragen:

2.1. **Wieviele Module** gehören in diesem Moment zum Programm Test12 und **wie heissen diese Module?**

2.2. **Wieviele int-Variablen** existieren in diesem Moment und **wie heissen diese int-Variablen?**
Geben Sie die **vollen Namen** der Variablen an (z.B. **Moses.verena** für eine Variable namens **verena**, die in einem Modul namens **Moses** "lebt").

Aufgabe 3 (20 Punkte): Was wird zum Bildschirm ausgegeben? Geben Sie für jede Befehlsfolge an, welchen Text sie zum Bildschirm ausgibt:

```
1 // Teilaufgabe 3.1.: -----
2 int sum=0;
3 for (int i=-12; i<+12; i+=5) sum += i;
4 System.out.println("sum: " + sum);

1 // Teilaufgabe 3.2.: -----
2 int anz=0;
3 for (int i=-3; i<=+5; i++) {
4     for (int j=+4; j>=-2; j--) {
5         anz++;
6     }
7 }
8 System.out.println("anz: " + anz);

1 // Teilaufgabe 3.3.: -----
2 int zahl=17, log=0;
3 while (zahl > 0) {
4     log++;
5     zahl /= 2;
6 }
7 System.out.println("log: " + log);
```

```

1      // Teilaufgabe 3.4.: -----
2      int oft=3, n=3;
3      do {
4          switch (n) {
5              case 1: oft++; n=4; break;
6              case 2: oft++; n=0; break;
7              case 3: oft++; n=5; break;
8              case 4: oft++; n=2; break;
9              case 5: oft++; n=1; break;
10         }
11     } while (n > 0);
12     System.out.println("oft: " + oft);

1      // Teilaufgabe 3.5.: -----
2      int enn = 17;
3      for (int i=1; i<6; i++) {
4          if (enn % 2 == 0) {
5              enn /= 2;
6          } else {
7              enn = 3 * enn + 1;
8          }
9      }
10     System.out.println("enn: " + enn);

```

Aufgabe 4 (20 Punkte): Programmieren Sie eine Funktion namens **sindDisjunkt** entsprechend dem folgenden „Skelett“:

```

1      public static boolean sindDisjunkt(int[] ir1, int[] ir2) {
2          // Verlaesst sich darauf, dass ir1 und ir2 Mengen von Ganzzahlen sind
3          // ("keine Doppelten"). Liefert true, wenn die Mengen ir1 und ir2
4          // disjunkt sind (d.h. wenn kein Element gleichzeitig in ir1 und in
5          // ir2 enthalten ist), und liefert sonst false.
6          // Null-Referenzen werden wie leere Mengen behandelt.
7          ...
8      }

```

Aufgabe 5 (15 Punkte): Jede Ganzzahl **n** (ungleich 0) hat genau zwei **unechte Teiler** (nämlich **1** und **n**) und ausserdem 0 oder mehr **echte Teiler**. Z.B. hat die Ganzzahl 12 ausser den beiden **unechten Teilern** 1 und 12 auch die vier **echten Teiler** 2, 3, 4 und 6. Eine Primzahl wie z.B. 17 hat nur die beiden unechten Teiler 1 und 17 (und keine echten Teiler, d.h. 17 hat 0 echte Teiler). Programmieren Sie eine Funktion namens **anzahlEchteTeiler** entsprechend dem folgenden „Skelett“:

```

1      public static int anzahlEchteTeiler(int n) {
2          // Verlaesst sich darauf, dass n ungleich 0 ist.
3          // Liefert die Anzahl der echten Teiler der Ganzzahl n.
4          ...
5      }

```

Beispiele für Aufrufe der Funktion **anzahlEchteTeiler**:

Der Aufruf **anzahlEchteTeiler(12)** soll **4** als Ergebnis liefern.

Der Aufruf **anzahlEchteTeiler(17)** soll **0** als Ergebnis liefern.

Lösung 4 (20 Punkte): Programmieren Sie eine Funktion namens **sindDisjunkt**:

```
1  public static boolean sindDisjunkt(int[] ir1, int[] ir2) {
2      // Verlaesst sich darauf, dass ir1 und ir2 Mengen von Ganzzahlen sind
3      // ("keine Doppelten"). Liefert true, wenn die Mengen ir1 und ir2
4      // disjunkt sind (d.h. wenn kein Element gleichzeitig in ir1 und in
5      // ir2 enthalten ist), und liefert sonst false.
6      // Null-Referenzen werden wie leere Mengen behandelt.
7
8      if (ir1 == null || ir2 == null) return true;
9      for (int i1=0; i1<ir1.length; i1++) {
10         for (int i2=0; i2<ir2.length; i2++) {
11             if (ir1[i1] == ir2[i2]) return false;
12         }
13     }
14     return true;
15 } // sindDisjunkt
```

Lösung 5 (15 Punkte): Jede Ganzzahl **n** (ungleich 0) hat genau zwei **unechte Teiler** (nämlich **1** und **n**) und ausserdem 0 oder mehr **echte Teiler**. Z.B. hat die Ganzzahl 12 ausser den beiden **unechten Teilern** 1 und 12 auch die vier **echten Teiler** 2, 3, 4 und 6. Eine Primzahl wie z.B. 17 hat nur die beiden unechten Teiler 1 und 17 (und keine echten Teiler, d.h. 17 hat 0 echte Teiler).

Programmieren Sie eine Funktion namens **anzahlEchteTeiler** :

```
1  public static int anzahlEchteTeiler(int n) {
2      // Verlaesst sich darauf, dass n ungleich 0 ist.
3      // Liefert die Anzahl der echten Teiler der Ganzzahl n.
4
5      // n = Math.abs(n); // So berechnet man den absoluten Betrag von n
6      if (n < 0) n = -n; // So geht es auch ohne die Funktion abs()!
7      int anz = 0; // Das Ergebnis dieser Funktion
8      for (int i=2; i<n; i++) {
9          if (n % i == 0) anz++;
10     }
11     return anz;
12 } // anzahlEchteTeiler
```