Institut für Informatik
Heinrich-Heine-Universität Düsseldorf
Prof. Dr. Alice McHardy
Robin Küppers
Sebastian Konietzny
Aaron Weimann

WS 2013/2014
12.11.2013
Übungsblatt 3

Einführung in die Informatik I
Übungen zur Vorlesung
Musterlösung

Die Lösungen der zu bearbeitenden Aufgaben müssen spätestens bis **Dienstag**, den **19.11.2013, 11:59 Uhr** über https://ilias.uni-duesseldorf.de abgegeben werden. Bitte achten Sie immer auf einen vollständigen und nachvollziehbaren Lösungsweg, da ansonsten Punkte abgezogen werden können! Vergessen Sie bitte nicht auf Ihrer Abgabe Ihren Namen und Ihre Matrikelnummer zu vermerken (bei Gruppenabgaben müssen sämtliche Mitglieder der Gruppe aufgeführt werden).

Hinweis: Es ist natürlich gestattet diese Aufgaben mittels der *Java*-Umgebung zu lösen, bedenken Sie aber, dass in der Klausur keinerlei elektronische Hilfsmittel zugelassen sind.

Kurze Erinnerung:

Sei `String s = "Info1"` eine Zeichenkette. Dann gibt `System.out.print(s)` die Zeichenkette „Info1“ auf der Konsole aus. Die Methode `System.out.println(s)` hingegen gibt die Zeichenkette „Info1“ mit anschließendem Zeilenumbruch aus.

3.1 Variablen und Literale (5 Punkte)

Beschreiben die nachfolgenden Beispiele gültige Variablenzuweisungen in *Java*? Falls ja, geben Sie den Wert der Variable an. Begründen Sie ihre Antwort.

1. `int a = 2;`
2. `int b = "2";`
3. ```
   int c;
   c = 2 / 2.0;
   ```
   ```

4.
```
String d;
d = 2.0f + " / 2";
```

5.
```
int e;
int f;
int g;
e = 10;
f = 3;
g = (e / f) * g;
```

Musterlösung

1. gültige Zuweisung: a hat den Integer-Wert 2.
2. ungültige Zuweisung: b ist nicht vom Typ String.
3. ungültige Zuweisung: $2/2.0 = 1.0$ ist vom Typ Double.
4. gültige Zuweisung: der String d hat den Wert `"2.0 / 2"`.
5. ungültige Zuweisung: g wird nie initialisiert.

3.2 Arrays (5 Punkte)

Seien die folgenden *Java*-Arrays gegeben:

```
int[] a = new int[] { 1, 2, 3, 4, 5, 6 };
int[][] b = new int[][] { { 0, 2, 4, 6, 8 }, {1, 3, 5, 7, 9 } };
int[][] c = new int[][] { { 1 }, { 2, 3 }, { 4, 5, 6 } };
```

Geben Sie in jeder Teilaufgabe eine Schleifenkonstruktion in *Java* an, die durch geeignetes Durchlaufen des gefragten Arrays die gegebene Ausgabe erzeugt. Zum Lösen jeder Teilaufgabe darf nur je eine äußere Schleife benutzt werden, aber das beliebige Verschachteln von Schleifen ist erlaubt. Für die Ausgabe sind nur `System.out.print()` und `System.out.println()` zugelassen.

1. Array: a
   Ausgabe:

   ```
   123456
   ```

2. Array: a
   Ausgabe:

   ```
   654321
   ```

3. Array: a
   Ausgabe:

   ```
   123321
   ```

4. Array: b
   Ausgabe:

   ```
   0123456789
   ```

5. Array: c
   Ausgabe:

   ```
   1
   23
   456
   ```

Musterlösung

1. ```
   for (int i = 0; i < a.length; i++) {
       System.out.print(a[i]);
   }
   ```

2. ```
   for (int i = a.length - 1; i >= 0; i--) {
       System.out.print(a[i]);
   }
   ```

3. ```
   for(int i = 0; i < a.length; i++) {
       if(i < 3)
           System.out.print(a[i]);
       else
           System.out.print(a[a.length - i - 1]);
   }
   ```

4. ```
   for (int i = 0; i < 10; i++) {
       System.out.print(b[i % 2][i / 2]);
   }
   ```

5. ```
   for (int i = 0; i < c.length; i++) {
       for (int j = 0; j < c[i].length; j++) {
           System.out.print(c[i][j]);
       }
       System.out.println();
   }
   ```
   ```

3.3 Schleifen und Bedingungen (5 Punkte)

Wie oft werden die folgenden Schleifen durchlaufen und welche Ausgaben erzeugen sie? Begründen Sie Ihre Antwort. Anmerkung: ein Schleifendurchlauf ist erst dann beendet, wenn alle Instruktionen innerhalb der Schleife ausgeführt wurden.

1. ```
   boolean b = false;
   while(!b) {
       System.out.println(b);
       b = !b;
   }
   ```

2. ```
   for(int i = 0; i < 3; i++) {
       for(int j = 0; j < 3; j++) {
       }
   }
   ```

3. ```
   for(int i = 1; i < 4; i++) {
       for(int j = 4; j > i; j--) {
           System.out.println("i * j = " + i * j);
       }
   }
   ```

4. ```
   int i = 0;
   do {
       i = i + 1000000000;
       if(i % 2 == 0) {
           System.out.println(i);
       }
   } while(i > 0);
   ```

5. ```
   int x[] = new int[1];
   while(true) {
       System.out.println(x[1]);
   }
   ```
   ```

Musterlösung

1. Durchläufe: 1, Ausgabe:

   ```
   false
   ```

2. Durchläufe: $3 \cdot 3 = 9$, Ausgabe: keine

3. Durchläufe: die äußere while-Schleife macht 3 Durchläufe (dann gilt $i = 4$). Die innere Schleife ist abhängig von der Zählervariable i der äußeren Schleife:

   1. Durchlauf (außen): $i = 1, j = 4$, Durchläufe (innen): $i - j = 4 - 1 = 3$
   2. Durchlauf (außen): $i = 2, j = 4$, Durchläufe (innen): $i - j = 4 - 2 = 2$
   3. Durchlauf (außen): $i = 3, j = 4$, Durchläufe (innen): $i - j = 4 - 3 = 1$

   $\Rightarrow 3 + 2 + 1 = 6$ Durchläufe insgesamt.

   Ausgabe:

   ```
   i * j = 4
   i * j = 3
   i * j = 2
   i * j = 8
   i * j = 6
   i * j = 12
   ```

4. Durchläufe: 3, da i nach 3 Durchläufen den positiven Integer-Wertebereich $2^{31}-1$ verlassen hat[1]. Zu Beginn der 3. Iteration gilt noch $i > 0$. Dann wird der positive Wertebereich um den Betrag $y = |(2^{31} - 1) - (3 \cdot 1000000000)| = |-852516353| = 852516353$ überschritten. Beim Hinzuaddieren von 1 zu $2^{31} - 1$ kommt es zu einem Überlauf des Wertebereiches in Java mit dem Wert -2^{31} als Ergebnis. Daher ist i in der 3. Iteration $i = -2^{31} + (852516353 - 1) = -1294967296$. Die innere Bedingung `i % 2 == 0` ist wahr für alle Werte von i. Da die Schleifenbedingung erst nach der Ausgabe ausgewertet wird, ist die Ausgabe:

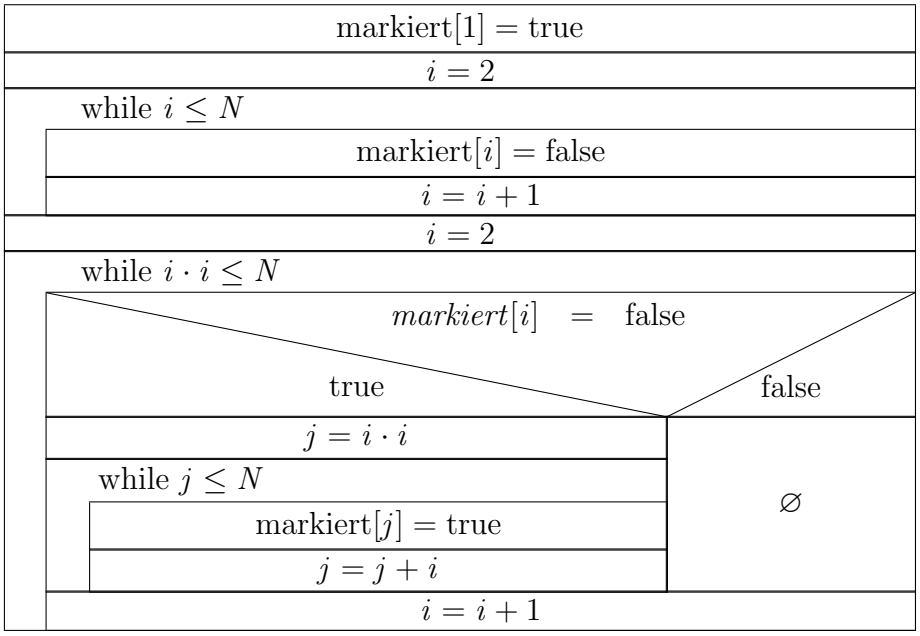
   ```
   1000000000
   2000000000
   -1294967296
   ```

5. Durchläufe: 0 (Schleife bricht während des ersten Durchlaufs mit einer `java.lang.ArrayIndexOutOfBoundsException` ab). Ausgabe: keine

[1] Eine ausführlichere Begründung ist NICHT notwendig, wird hier aber angegeben.

3.4 Java, Sieb des Eratosthenes (10 Punkte)

Auf Übungsblatt 2 haben Sie das Sieb des Eratosthenes als Pseudo-Code kennengelernt. Zur Erinnerung ist im Folgenden der Algorithmus als Struktogramm beschrieben:



Geben Sie den Algorithmus nun in *Java* an. Ihr *Java*-Programm soll den Parameter N über das String-Array `args[]` der `main`-Methode übergeben bekommen. Die Ausgabe des Arrays `markiert` soll über die Kommandozeile erfolgen. Durchlaufen Sie das Array `markiert` in einer Schleife und geben Sie jedes Element auf der Kommandozeile aus. Hinweis: Die abgegebene Lösung muss ein lauffähiges *Java*-Programm sein.

1. Implementieren Sie das Sieb des Eratosthenes in `Java` unter exklusiver Verwendung von while-Schleifen.

2. Verändern Sie ihre Implementierung aus Teilaufgabe 1 so, dass alle while-Schleifen durch äquivalente for-Schleifen ersetzt werden.

3. Welche Variante halten Sie für eleganter? Begründen Sie ihre Meinung.

Musterlösung

1.
```
public static void main(String[] args) {

    int N = Integer.parseInt(args[0]);

    boolean[] markiert = new boolean[N + 1];
    markiert[0] = true;
    markiert[1] = true;
    int i = 2;
    while (i <= N) {
        markiert[i] = false;
        i = i + 1;
    }

    i = 2;
    while (i * i <= N) {
        if (!markiert[i]) {
            int j = i * i;
            while (j <= N) {
                markiert[j] = true;
                j = j + i;
            }
        }
        i = i + 1;
    }

    System.out.println("Eingabe: N = " + N);
    System.out.println("Zahl\tnicht-prim");
    for (int j = 1; j < markiert.length; j++) {
        System.out.println(j + "\t" + markiert[j]);
    }

}
```

2.
```
public static void main(String[] args) {

    int N = Integer.parseInt(args[0]);

    boolean[] markiert = new boolean[N + 1];
    markiert[0] = true;
    markiert[1] = true;
    for (int i = 2; i <= N; i++) {
        markiert[i] = false;
    }

    for (int i = 2; i * i <= N; i++) {
        if (!markiert[i]) {
            for (int j = i * i; j <= N; j = j + i) {
                markiert[j] = true;
            }
        }
    }

    System.out.println("Eingabe: N = " + N);
    System.out.println("Zahl\tnicht-prim");
    for (int j = 1; j < markiert.length; j++) {
        System.out.println(j + "\t" + markiert[j]);
    }

}
```

3. Die 2. Variante (for-Schleifen) ist eleganter, da for-Schleifen für die Verarbeitung von Arrays kompakteren Quellcode ergeben, weil Laufvariablen innerhalb der Schleifendeklaration deklariert, initialisiert und inkrementiert werden.