Lösung zu Probeprüfung HS 16

Hilfsmittel

Beliebige schriftliche Unterlagen

Punktzahl

35

Zeit

35 min

Hinweis:

- Die Prüfung dauert nur 25 Minuten. Die Probeprüfung ist also eher zu gross.

Anweisung

- Sie können die Lösungen entweder schriftlich oder elektronisch erfassen.

Falls Sie schriftlich erfassen:

- Sie können die Lösungen direkt auf die Prüfungsblätter schreiben. Die Prüfungsblätter werden abgegeben. Falls nötig bekommen Sie von mir zusätzliche Prüfungsblätter.
- Falls Sie Zusatzblätter brauchen, benutzen Sie für jede Aufgabe separate Zusatzblätter.
- Sie können auch mit Bleistift schreiben.
- Benutzen Sie keinen roten Stift.

Abgabe

- Schreiben Sie Ihren Namen auf die Prüfungsblätter und alle Zusatzblätter.
- Geben Sie die Prüfung mit ihren schriftlichen Lösungen der Aufsichtsperson ab.
- Laden Sie Ihre elektronische Lösung auf den Memory Stick. Geben Sie den Memory Stick der Aufsichtsperson.

1. **(15 P.) Nebenläufigkeit/Arrays**. Gegeben ist die Klasse *List* wie folgt:

```
1  class List {
2    private String[] array;
3    private int size;
4
5    public List() {
6           array = new String[10];
7           size = 0;
8    }
9
10   // Add an String at the end of the list
11   public void add(String s) {
12          if (size >= array.length) {
13                grow();
14          }
15          array[size] = s;
16          size++;
17   }
18
19   // Remove the string at the end of the list
20   public String remove() {
21          if (size == 0) {
22                return null;
23          }
24          size--;
25          String result = array[size];
26          array[size] = null;
27          return result;
28   }
29
30   // Get the string at given position
31   public String itemAt(int index) {
32          if (index < 0 || index >= size) {
33                return null;
34          } else {
35                return array[index];
36          }
37   }
38
39   // Get the list size
40   public int size() {
41          return size;
42   }
43
44   private void grow() {
45          String[] newArray = new String[array.length * 2];
46          for (int i=0; i<size; i++) {
47                newArray[i]=array[i];
48          }
49          array = newArray;
50   }
51 }
```

Folgendes Beispiel zeigt, wie die Klasse verwendet werden kann:

```
52 public class Beispiel1 {
53   public static void main(String[] args) {
54          List list = new List();
55          list.add("Fritz");
56          list.add("Müller");
57          System.out.println(list.size()); //     Erwartet: 2
58          System.out.println(list.itemAt(1));     // Erwartet: Müller
59          System.out.println(list.remove());      // Erwartet: Müller
60          System.out.println(list.remove());      // Erwartet: Fritz
61   }
62 }
```

a) (10 P.) Skizzieren Sie ein Szenario, wo es durch nebenläufiges Ausführen von zwei Threads zu einem Absturz des Programmes kommen kann.

Beschreiben Sie konkret, wie das Array aussieht und welche Methoden der Klasse *List* die Threads benutzen.

Variante 1: Zwei Threads sind gleichzeitig in der Methode add

Annahme:

- Die Liste hat die Grösse 10.
- Momentan sind 9 Elemente in der Liste (size == 9)

Thread 1: Ruft die Methode add(„Hello“) auf

- Wird vor der Zeile 15 unterbrochen. Da 9 <= 10, wurde noch kein grow gemacht.
-

Thread 2: Ruft die Methode add(„World“) auf.

- Läuft durch die Methode add durch. Es wird ebenfalls kein grow gemacht, aber die Variable size wird auf 10 gesetzt.

Thread 1: läuft weiter

- Zeile 15 führt nun folgende Anweisung aus:
  array[10] = „Hello“;
  Dies führt zu einer ArrayIndexOutOfBoundsException.

Variante 2: Zwei Programme machen gleichzeitig ein remove.

…

b) (5 P.) Machen Sie die Klasse *List* thread-safe!

Beschreiben Sie die Änderungen (inkl. Angabe der Zeilennummer), um die Klasse thread-safe zu machen.

Anforderungen:

- Ihre Änderungen müssen möglichst viel Nebenläufigkeit zulassen.
- Ihre Änderungen sollen möglichst wenig Overhead, beispielsweise Synchronisation, haben.

Es braucht ein synchronized an folgenden Stellen:

Zeile 11: add

Zeile 20: remove

Zeile 31: itemAt (darüber lässt sich streiten)

Es braucht **kein** synchronized an folgenden Stellen:

Zeile 5: Konstruktor

Zeile 40: size

Zeile 44: grow (grow ist ja von aussen nicht sichtbar und wird nur innerhalb der synchronisierten Methode add aufgerufen).

2. **(20 P.) Primzahlen**. Eine Zahl n ist eine Primzahl, falls im Bereich 2..n-1 kein Teiler (engl. Divisor) existiert.

   Im folgenden Codeausschnitt prüft die Methode `isPrim` gemäss der Definition, ob eine Zahl eine Primzahl ist. Durch die Nutzung von Threads soll die Prüfung performanter erfolgen. Dazu muss der Zahlenbereich 2..n-1 in gleich grosse Bereiche aufgeteilt werden und diese separat auf Teiler geprüft werden. Die Methode `isPrimWithRanges` tut genau dies, nur nutzt Sie noch keine Threads.

   Schreiben Sie eine Methode `isPrimWithThreads`, die analog zu `isPrimWithRanges` arbeitet, aber Threads benutzt. Leiten Sie dazu eine Klasse `DivisorThread` von Thread ab. Geben beim Konstruktor und den zu bearbeitenden Bereich mit. Die Klasse `DivisorThread` soll in einer Instanzvariablen abspeichern, ob im Bereich ein Teiler existiert oder nicht. Aus der Methode `isPrimWithThreads` können Sie diese Instanzvariablen abfragen.

```
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) throws Exception {
        int number = 17;

        // check that the number is prime
        System.out.println("Is " + number + " prime? " + isPrim(number));

        // check that the number is prime
        System.out.println("Is " + number + " prime? " + isPrimWithRanges(number));
    }

    public static boolean isPrim(int number) {
        // check that number has no divisor in the range from 2 to number-1
        for (int i=2; i<number; i++) {
            if (number % i==0) {
                return false;
            }
        }
        return true;
    }

    public static boolean isPrimWithRanges(int number) {
        boolean isPrim = true;
        int noRanges = 4;
        // the range 2 to number-1 has number-2 numbers
        int noNumberInRange = roundUp(number-2, noRanges);
        int from=2;
        for (int i=0; i<noRanges; i++) {
            int to = Math.min(from+noNumberInRange-1, number-1);
            boolean hasDivisor =hasDivisior(number, from, to);
            System.out.println("\tRange " + from + "-"+ to + ": " + hasDivisor);
            from = to + 1;
            isPrim = isPrim && !hasDivisor;
        }

        return isPrim;
    }

    private static boolean hasDivisior(int number, int from, int to) {
        for (int i=from; i<=to; i++) {
            if (number % i==0) {
                return true;
            }
        }
        return false;
    }


    private static int roundUp(int num, int divisor) {
        return (num + divisor - 1) / divisor;
    }
}
```

```
Siehe Code
```