Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A.B. Cremers

Aufgabenblatt 8

— Veröffentlicht am 23.06.2010 —

Abgabe bis 29.06.2010, 13:00 per SVN

Aufgabe 1 *(Null-Objekt, Ausnahmebehandlung, Nebenläufige Programmfäden - 2 Punkte)*

Überlegen Sie sich für jede Aussage, ob sie wahr oder falsch ist und kreuzen Sie das entsprechende Feld an. Jede richtige Antwort ergibt 0,5 Punkte, jede Falsche -0,5 Punkte und jede Fehlende 0 Punkte. Also: Besser nicht ankreuzen, statt falsch ankreuzen!

wahr	falsch	
		Jeder Java-Variable kann man das spezielle „Objekt“ `null` zuweisen. Damit wird eine vorher vorhandene Referenz auf ein Objekt im Speicher gelöscht.
		Einem einzelnen `try`-Block können mehrere `catch`-Blöcke folgen.
		Einem `try-catch`-Block muss immer ein `finally`-Block folgen.
		Programmcode, der in Java nebenläufig ausgeführt werden soll, muss in Unterklassen der Klasse `Thread` implementiert werden.

Aufgabe 2 *(Ausnahmebehandlung - 8 Punkte)*

Betrachten Sie das folgende Programm `TestTrace`, welches die Methode `methodA()` der Klasse `CallEg` aufruft:

```
class CallEg {
  public void methodA() throws ArithmeticException { }
  public void methodB() throws ArithmeticException { }
  public void methodC() throws ArithmeticException { }
}

public class TestTrace {
  public static void main(String[] args) {
    CallEg eg = new CallEg(); // use default constructor
    try {
      eg.methodA();
    } catch (ArithmeticException oops) {
      oops.printStackTrace();
    }
  }
}
```

a) Der `catch{}`-Block des Hauptprogramms gibt den *Stacktrace* der abgefangenen Ausnahme aus. Ergänzen Sie `methodA()` so, dass eine Division durch Null auftritt. Betrachten Sie die Ausgabe und verfolgen Sie die angegebenen Aufrufe im Stack.

b) Verschieben Sie die Division aus `methodA()` nach `methodC()`. Ändern Sie den Code so, dass `methodA()` die Methode `methodB()` aufruft, welche wiederum `methodC()` aufruft. Führen Sie das Programm aus und beobachten Sie, wie sich die Ausgabe ändert.

c) Ändern Sie den Code aus Aufgabenteil b) so, dass `methodB()` von `methodA()` innerhalb eines `try{}`-Blocks aufgerufen wird, ebenso soll `methodC()` von `methodB()` in einem `try{}`-Block aufgerufen werden. Setzen Sie auch die Division in `methodC()` in einen `try{}`-Block. Setzen Sie hinter jeden `try{}`-Block ein `catch{}`, welches die Ausnahme fängt, einen *Stacktrace* ausgibt und die Ausnahme an seinen Aufrufer wirft. Kommentieren Sie die Ausgabe.

Aufgabe 3 *(Garbage Collection - 5 Punkte)*

Beschreiben Sie Aufgabe und Funktionsweise des *Java Garbage Collector*. Worin liegen die Vorteile gegenüber anderen Programmiersprache ohne *Garbage Collector* (wie zum Beispiel C++), worin die Nachteile? Wie können Sie die *Garbage Collection* programmiertechnisch unterstützen?

Aufgabe 4 *(Threads - 6 Punkte)*

Schreiben Sie eine Klasse `DateiBeobachter` mit Konstruktoren `DateiBeobachter(String)` und `DateiBeobachter(java.io.File)`, die das Interface `Runnable` implementiert. Die Klasse soll beobachten, ob sich eine Datei im Dateisystem ändert, und gegebenenfalls eine entsprechende Meldung ausgeben.[1] Lassen Sie sich während der Programmausführung im Abstand von einer halben Sekunde den Namen der Datei anzeigen. Legen Sie zum Testen eine Textdatei an und modifizieren Sie diese schließlich zur Laufzeit des Threads!

[1] Benutzen Sie die Methode `java.io.File.lastModified()` um auf die Zeit der letzten Modifikation zuzugreifen.