Arbeitsblatt: Threads beenden

Letzte Woche haben Sie gesehen, wie man mittels `interrupt` einen Thread beenden kann. In diesem Arbeitsblatt betrachten Sie einen einfachen Mechanismus zum kooperativen Beenden eines Threads (ohne die `interrupt` Methoden zu verwenden). Auf dem AD finden Sie das Archiv 04_LE_JMM.zip. Es enthält ein Eclipse Java Projekt, das die relevanten Klassen bereitstellt.

Im Package `jmm.stop` finden Sie folgende Klasse:

```
static class StoppableThread extends Thread {

	private boolean running = true;

	public void terminate() { running = false; }

	public boolean isRunning() { return running;  }

	@Override
	public void run() {
		while (isRunning()) {
			doSomeWork();
		}
	}

	private void doSomeWork() {
		for (int i = 0; i < 10; i++) {}
	}
}
```

Die Idee ist, dass `StoppableThread` in der `run()` Methode solange `doSomeWork()` aufruft, bis ein anderer Thread seine `terminate()` Methode aufruft. Dann soll die while-Schleife beendet werden und der Thread terminieren.

Die Klasse definiert dazu folgende Methoden:

- `terminate()` um dem Thread mitzuteilen, dass er bei der nächsten Gelegenheit stoppen soll.
- `isRunning()` um den Thread zu fragen, ob er weiterlaufen soll. `isRunning()` gibt `true` zurück bis jemand `terminate()` aufgerufen hat. Ab diesem Zeitpunkt wird `false` zurückgegeben.

Aufgaben:

1. In der main Methode ist ein „Test"-Programm vorbereitet. Zuerst wird darin ein `StoppableThread` erzeugt, für eine Sekunde laufen gelassen und dann mit einem Aufruf von `terminate()` beendigt. Welche Ausgabe erwarten Sie? Wenn Sie das Programm ausführen, stimmt das Resultat mit Ihren Erwartungen überein?
   Falls **Ja** => Setzen Sie sich zu einem Kollegen und beginnen Sie nochmals mit Aufgabe 1.
   Falls **Nein** => Sie sind nun leicht irritiert. Weiter mit Aufgabe 2.

2. Sie haben festgestellt, dass der `StoppableThread` sich eher wie ein „UnstoppableThread" verhält. Verwenden Sie Konsolenausgaben um zu schauen, wieso die while Schleife nicht terminiert. Geben Sie z.B. die Antwort von isRunning() innerhalb der while Schleife aus und führen Sie das Testprogramm nochmals aus.
   Entsprechen die Ausgaben Ihren Erwartungen? Hat auf einmal alles funktioniert?
   Falls beides **Ja** => Macht die JVM einfach was Sie will? Sie wurden gewarnt: Concurrent Programming ist sehr trickreich! Sie werden aber heute das Java Memory Model (JMM) kennenlernen und damit das Verhalten erklären können. Weiter mit Aufgabe 3.
   Falls eines **Nein** => Bitte zeigen Sie uns Ihr Programm. Wir werden dann gemeinsam weiterschauen.

3. Experimentieren Sie mit Synchronisation. Entfernen Sie zuerst die Konsolenausgaben wieder und markieren Sie dann die `terminate()` und/oder die `isRunning` Methode mit dem synchronized Keyword und lassen Sie das Programm nochmals laufen. Notieren Sie sich die Resultate:

isRunning()	terminate()	Programm terminiert?
-	-	
-	synch	
synch	-	
synch	synch	

4. (Optional) Entfernen Sie die synchronized Keywords und markieren Sie die boolean Variable running als `volatile`. Terminiert das Programm jetzt?

5. (Optional) Entfernen Sie die volatile-Deklaration aus der vorhergehenden Aufgabe wieder und lassen Sie das Programm im Debugger laufen. Terminiert das Programm?
   Falls **Ja** => Lesen Sie den Wikipedia-Eintrag http://en.wikipedia.org/wiki/Heisenbug
   Falls **Nein** => Dann halten Sie den Thread an und führen Sie das Programm mit Einzelschritten weiter und/oder lesen Sie den Wert der Variablen `running` aus und lassen dann das Programm weiterlaufen.