Übungsblatt 5

Aufgabe 13 Multi-Threading Einstieg

Erstellen Sie ein Programm, welches neben dem Hauptthread einen weiteren Thread laufen läßt. In diesem Thread soll ein Countdown von 10 Sekunden laufen.

- Der Countdown soll jede Sekunde die verbleibende Zeit ausgeben
- Die Eingabe im Hauptthread soll währenddessen aktiv sein
- Der Hauptthread soll nach Ablauf des Countdowns geschlossen werden

Die Technologie, über die Multi-Threading implementiert wird, ist hierbei frei wählbar.

Aufgabe 14 Thread-Synchronisierung

Wieder soll ein Programm geschrieben werden, welches Multi-Threading beherrscht. Nach Eingabe einer beliebigen (in der Regel sehr großen) Zahl N, sollen sich P Threads an die Aufgabe machen, eine gemeinsame globale Variable σ zu inkrementieren.

- σ soll Anfangs den Wert 0 haben und als vorzeichenlose, 64-bit Ganzzahl angelegt werden
- P ist ebenfalls frei wählbar, mit $P \in [1, N]$

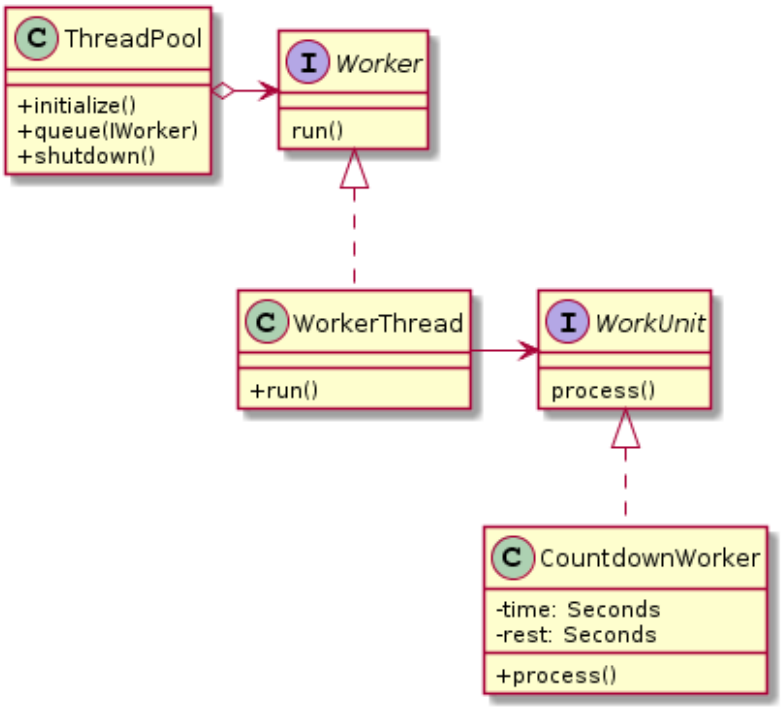
Das Ergebnis muss am Ende überprüft werden. Für ein korrektes Ergebnis muss $\sigma = N$ gelten. Folgende Fragen sollten durch das Programm beantwortet werden können:

- Stimmt das Ergebnis wenn ohne Synchronisierung gearbeitet wird?
- Wieviel Zeit benötigt das Programm mit Synchronisierung gegenüber der Version ohne?
- Wie könnte man den Overhead der Synchronisierung auf ein Minimum reduzieren, ohne dabei die Zuverläßigkeit einzuschränken?

Zeichnen Sie ein Aktivitätsdiagramm, welches Ihren verbesserten Algorithmus für 2 Threads darstellt.

Aufgabe 15 Ein einfacher Thread-Pool

Schreiben Sie eine Klasse *MyThreadPool* die verschiedene Threads laufen läßt, welche nur darauf warten mit Instanzen von *WorkUnit* (siehe folgendes UML Diagramm) versorgt zu werden.



Zeichnen Sie ein Objektdiagramm für das folgende Szenario:

- Ein vorhandener, aktiver *ThreadPool*
- 2 Threads vorhanden
- 3 konkrete *WorkUnit* Objekte, davon 2 gerade in Behandlung und einer in der Warteschlange

Ihr Programm soll so funktionieren, dass in einem Hauptthread Zahlen eingegeben werden können, welche die Laufzeit einer neuen *WorkUnit* in Sekunden angibt.
Der *ThreadPool* soll jede Aktion mit entsprechender Ausgabe versehen (z.B. *WorkUnit* in Warteschlange gesetzt, *WorkUnit* beendet). Die Anzahl der Threads, die im *ThreadPool* laufen, soll einstellbar sein.

! Wichtig

Alle Diagramme können über Anwendungen (z.B. PlantUML, yUML, Visual Studio, ...) oder auch per Hand gezeichnet werden.