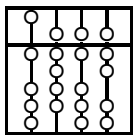
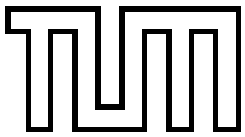


TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



Lehrstuhl für Sprachen und Beschreibungsstrukturen SoS 2007
Praktikum Grundlagen der Programmierung Aufgabenblatt 9
Prof. Dr. Helmut Seidl 2. Juli 2007

Abgabe: Montag 12:00, KW 28 beim jeweiligen Tutor per Mail

Praktikum Grundlagen der Programmierung

Aufgabe 40 Siedler – Verklemmung

In dieser Aufgabe soll der Umgang mit den einfacheren Konzepten der Nebenläufigkeit vorgeführt werden.

a) Implementieren Sie die Klasse `Siedler` als Kindklasse der Klasse `Thread`. Statten Sie Ihren Siedler mit einem Konstruktor aus, der die Übergabe von zwei Monitorobjekten als erwünschte Rohstoffe erlaubt.
b) Erweitern Sie ihren `Siedler` um eine `run`-Methode, in der Sie zuerst ein Lock auf den ersten Rohstoff beantragen, dann 2 Sekunden warten, und dann ein Lock auf den zweiten Rohstoff beantragen und wieder 2 Sekunden warten, bevor Sie beide Locks wieder aufgeben. Achten Sie auf ausreichend viele erläuternde Statusmeldungen Ihrer `run()`-Methode!
c) Testen Sie Ihre `Siedler`-klasse mit den zwei Siedlern *Hugo* und *Helga*, sowie den Rohstoffen *Lehm* und *Holz* in einem `main()`-Programm und untersuchen Sie, ob und unter welchen Umständen es zu einer Verklemmung kommen kann.

Aufgabe 41 Parallele Matrixmultiplikation

Nutzen Sie das Java Thread-Konzept zur parallelen Multiplikation von Matrizen. Die Multiplikation zweier Matrizen ist folgendermaßen definiert:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{n2} & \dots & c_{mk} \end{pmatrix}$$

wobei die Einträge c_{ij} gegeben sind durch

$$c_{ij} = \sum_{l=1,\dots,n} a_{il} \cdot b_{lj}.$$

Implementieren Sie eine Klasse `RowColumnMult` als Erweiterung der Klasse `Thread`, deren Instanzen genau ein Element der Ergebnismatrix berechnen.

Dem Konstruktor werden die zu multiplizierenden Matrizen A und B, die Ergebnismatrix C sowie die Zeile und Spalte des zu berechnenden Elements übergeben. In der Methode `run()` wird das gewünschte Ergebniselement berechnet und in die Ergebnismatrix C geschrieben.

Fügen Sie zu der Klasse `RowColumnMult` eine `main`-Methode hinzu, in der Sie exemplarisch zwei Matrizen A und B erzeugen. Berechnen Sie alle Elemente des Produktes AB *parallel*, indem sie

für jedes Ergebniselement einen Thread der Klasse `RowColumnMult` erzeugen und starten. Geben Sie das Multiplikations-Ergebnis auf dem Bildschirm aus.

Aufgabe 42 (H) Aussichtspunkt **(10 Punkte)**

Auf einem Fernsehturm gibt es eine Aussichtsplattform, auf der maximal 10 Leute Platz haben. Vom Erdboden bis zur Aussichtsplattform führt ein Lift, für eine einzelne Person. Der Lift soll jederzeit Leute von der Aussichtsplattform zum Erdboden transportieren dürfen, in die Gegenrichtung jedoch maximal so lange, bis die Aussichtsplattform voll besetzt ist. In diesem Fall müssen Touristen, die die Plattform besuchen möchten unten vor dem Aufzug warten. Implementieren Sie:

a) Eine Klasse `Aufzug`, die einzelne Personen vom Erdboden zur Plattform und umgekehrt transportiert. Der Aufzug ist dabei mit einem Mechanismus zu versehen, der garantiert, dass sich nie mehr als 10 Leute auf der Plattform aufhalten können.
b) Eine Klasse `Tourist`, die einen Aufzug betritt um eine zufällige Zeitspanne auf der Aussichtsplattform des Turms zu verweilen, und dann denn Aufzug ruft, um den Turm wieder zu verlassen

Simulieren Sie die einzelnen auftretenden Aktionen mithilfe von Textausgaben und `sleep()` Anweisungen. Testen Sie Ihre Simulation ausgiebig!

Aufgabe 43 (H) MiniJVM-Interpreter **(15 Punkte)**

Aus der Vorlesung kennen sie die VAM, den grafischen MiniJVM-Programm Interpreter. In dieser Aufgabe sollen Sie selbst einen MiniJVM-Programm Interpreter entwerfen und schreiben. Zur Lösung sollen bisher erlernte Konzepte wie Vererbung, IO, Exceptions und Generics möglichst sinnvoll eingesetzt werden.

Zur Ausführung der MiniJVM Befehle sollten Sie eine virtuelle Maschine entwerfen. Diese Maschine braucht als Komponenten einen Instruktionszähler, der immer auf den nächsten auszuführenden Befehl zeigt, sowie einen Stack, in dem lokale Variablen und temporäre Werte abgelegt werden können. Als Eingabe und Ausgabe können Sie MiniJava-Dialoge oder die `System.out/System.in` Klassen nutzen!

a) Die virtuelle Maschine muß Listen von MiniJVM-Befehlen abarbeiten können. Jeder Mini-JVM Befehl muß daher die Methode **`void`** `execute()` bereit stellen. Für jeden Befehl der MiniJVM muß also eine eigene Klasse erstellt werden, die jeweils einen MiniJVM Befehl darstellen kann.
b) Der Mini-JVM Interpreter soll seine Befehle aus `.jvm` Dateien einlesen können, und diese in einer Liste ablegen, die er dann an die Interpreterklasse zur Ausführung weitergeben muß.
c) Die hier implementierte virtuelle Maschine soll *Debugging* von MiniJVM Programmen zulassen. Dazu soll die Maschine nach jedem Teilbefehl eine Pause machen, in der es möglich ist anzugeben, ob der letze Befehl *zurückgenommen* wird, oder mit dem nächsten Befehl *weitergemacht* wird.
   Implementieren Sie dazu als Gegenstück zur Befehlsliste einen *Undo-Stack*, auf den Sie eine Kopie des Befehls nach seiner Abarbeitung legen. Befehle sollten Sie außerdem mit einer **`void`** `undo()` Methode ausstatten, die diesen Befehl wieder rückgängig macht.