Übung 13: Threads und das Erzeuger-Verbraucher-Problem

In einem Restaurant arbeiten n Köche und m Bedienungen. Die Köche kochen was das Zeug hält und stellen die fertigen Speisen in eine Durchreiche. Vor der Durchreiche stehen die Kellner und warten. Immer wenn ein Essen fertig ist nimmt es ein Kellner und bringt es zum Gast.

Die Durchreiche hat eine Kapazität von k Essen, d.h. sobald dort k oder mehr Essen stehen, muss ein Koch mit seinem fertig gekochten Essen warten, bis der nächste Kellner ein Essen weggenommen hat und er seines hineinstellen kann.

Aufgabe 1: Einfache (nicht-synchronisierte) Durchreiche Implementieren

Implementieren Sie das gegebene Interface Durchreiche:

```
public interface Durchreiche<T> {
    T get();   // Etwas aus Durchreiche holen
    void put(T o);  // Etwas in Durchreiche stellen
}
```

Hinweise:

- Eine Durchreiche soll eine maximale Anzahl "Stellplätze" haben.
- Verwenden Sie das Interface `java.util.Queue` sowie (z.B.) die Klasse `LinkedList` aus der Java Bibliothek; verwenden Sie keine Klassen aus dem Paket `java.util.concurrent`.

Aufgabe 2: Durchreiche synchronisieren

Die Durchreiche soll später von mehreren Köchen und Bedienungen gleichzeitig benutzt werden. Achten Sie daher auf korrekte Synchronisierung: Die Warteschlange (Queue) darf immer nur von einem Thread gleichzeitig benutzt werden. Oder anders ausgedrückt: Sowohl in `put` als auch `get` muss der Zugriff auf die Warteschlange in einem kritischen Abschnitt (`synchronized (...)`) erfolgen. Weiterhin gilt: Ist die Schlange voll, so muss in `put` gewartet werden (`wait()`) bis wieder Platz ist; ist die Schlange leer, so muss in `get` gewartet werden, bis wieder etwas da ist. Befindet man sich in einem kritischen Abschnitt, so kann auf dem "Schlüsselobjekt" jeweils `wait` bzw. `notifyAll` aufgerufen werden, um zu warten bzw. andere Threads aufzuwecken. Im Normalfall werden beim Einfügen bzw. Entnehmen immer alle anderen Threads aufgeweckt (es könnte ja jemand warten...).

Zur Erinnerung:

```
class Klasse {
    synchronized void methode1() {
        wait();  // schlafen legen, bis ein Anderer auf diesem Objekt notifyAll
aufruft
        notifyAll();  // andere Threads aufwecken, die aktuell auf dieses Objekt
warten
    }
    // oder äquivalent mit Schlüsselobjekt
    void methode2() {
        synchronized (this) {
            this.wait();
            this.notifyAll();
        }
    }
    // oder mit anderem Schlüsselobjekt
    Object key = new Object();
    void methode3() {
        synchronized (key) {
            key.wait();
            key.notifyAll();
        }
    }
}
```

Es ist wichtig sich zu merken, über welches Objekt gesichert wird (`this` oder ein anderes?), denn entsprechend muss bei diesem Objekt `wait` bzw. `notifyAll` aufgerufen werden.

Aufgabe 3: Kellner und Koch

Implementieren Sie einen Koch als `Runnable`, welcher einen Namen hat (`String`) sowie (nacheinander) eine vorgegebene Anzahl an Speisen erzeugt und in die `Durchreiche<Speise>` stellt. Implementieren Sie analog dazu eine Bedienung, die verständlicherweise nicht kocht, sondern eine gewisse maximale Zahl an Bedienvorgängen erledigt.

Hinweise:

- Aus dem obigen Text sollten Sie erschlossen haben, dass sowohl Koch als auch Bedienung jeweils einen `String` (name), einen `int`-Wert (Anzahl) sowie eine `Durchreiche<Speise>` im Konstruktor entgegen nehmen!
- Ein Koch soll eine gewisse (zufällige) Zeit warten, bevor er eine Speise in die Durchreiche stellt, um die Kochdauer zu simulieren; analog sollen Bedienungen nach dem Abholen etwas warten, um das Austragen zu simulieren. (z.B. `Thread.sleep((int) (Math.random() * 3000))`)
- Dokumentieren Sie die Aufrufe von `put` und `get` durch geeignete Ausgaben auf `System.out`, z.B. "`Speise <...> {in,aus} Durchreiche {gestellt,genommen}`", ähnlich dem Ausgabe-Beispiel unten.

Aufgabe 4: Programm

Schreiben Sie ein Programm namens Restaurant (also eine Klasse `Restaurant` mit einer `main` Methode), in dem zunächst eine Durchreiche erstellt wird, welche dann an Köche und Kellner weitergegeben wird. Variieren sie die Anzahl der Köche, Kellner, sowie Speisen und die Kapazität der Durchreiche.

- Kann es vorkommen, dass sich vor und hinter der Durchreiche jeweils eine Schlange bildet, d.h. sowohl Köche als auch Kellner stehen an?
- Muss man dies bei der Implementierung der Klasse Durchreiche beachten?
- Was wäre zu tun, um auch die Bestellvorgänge zu modellieren?

Hinweis: Starten Sie jeden Koch und jede Bedienung in einem separaten Thread. Das Programm wird terminieren, sobald alle Köche und Bedienungen zu ihrem jeweiligen Ende gekommen sind.

Eine mögliche Ausgabe könnte so aussehen:

```
Hans hat Essen 0 in die Durchreiche gestellt

Bernd hat Essen 0 aus der Durchreiche genommen.

Albert hat Essen 0 aus der Durchreiche genommen.

Peter hat Essen 0 in die Durchreiche gestellt

Peter hat Essen 1 in die Durchreiche gestellt

Gisela hat Essen 1 aus der Durchreiche genommen.

Hans hat Essen 1 in die Durchreiche gestellt

Bernd hat Essen 1 aus der Durchreiche genommen.

Peter hat Essen 2 in die Durchreiche gestellt

Albert hat Essen 2 aus der Durchreiche genommen.

Hans hat Essen 2 in die Durchreiche gestellt

Gisela hat Essen 2 aus der Durchreiche genommen.

Hans hat Essen 3 in die Durchreiche gestellt

Bernd hat Essen 3 aus der Durchreiche genommen.

Peter hat Essen 3 in die Durchreiche gestellt

Albert hat Essen 3 aus der Durchreiche genommen.

Peter hat Essen 4 in die Durchreiche gestellt

Gisela hat Essen 4 aus der Durchreiche genommen.

Peter hat Essen 5 in die Durchreiche gestellt

Gisela hat Essen 5 aus der Durchreiche genommen.
```