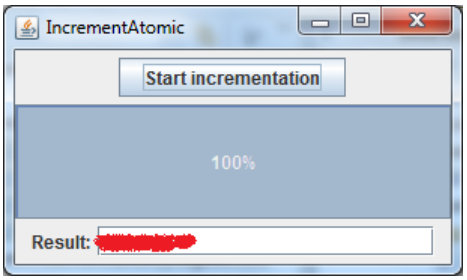
1. Eine Anweisung ist *atomar*, wenn diese vollständig und ohne durch einen anderen Thread unterbrochen zu werden durchgeführt wird. Sie sollen nun testen, ob die Anweisung `i++` atomar ist.

```
public class Int
{
   public int i = 0;
}
```

Definieren Sie sich dazu zu Testzwecken eine Klasse `Int`, welche in sich eine öffentliche Membervariable `int i` birgt. Definieren Sie sich dann weiters eine Klasse `Increment` – abgeleitet von `Thread` – welche in `run()` den Inhalt von `i` eine Million Mal inkrementiert. Dazu muss dieser Klasse bei ihrer Instanziierung das Objekt vom Typ `Int` übergeben werden.



Entwerfen Sie eine einfache *Benutzerschnittstelle*, anhand welcher das Inkrementieren *gestartet*, der *Fortschritt* und das *Ergebnis* des Inkrementierens erfragt werden können. Wenn das Inkrementieren gestartet wird, sollen *zwei Thread-Objekte* instanziiert werden, welche gleichzeitig die Variable `i` inkrementieren. Wenn am Ende der Abarbeitung in `i` der Wert `2.000.000` steht, so bedeutet dies, dass die Anweisung `i++` atomar ist und *nicht* von einem anderen Thread unterbrochen werden kann.

Damit beim Inkrementieren auch der *Fortschrittsbalken* angesprochen und zum Schluss das Ergebnis in das *Textfeld* geschrieben werden können, müssen diese beiden Objekte dem Thread ebenfalls bei der Instanziierung übergeben werden.

2. Insbesondere bei *Datenbankanwendungen* werden häufig *fortlaufende Nummerierungen* für Objekte benötigt (z. B. bei der Vergabe von Kunden- oder Rechnungsnummern). Die breitgestellte Klasse `SerializedCounter` stellt solche fortlaufenden Nummern zur Verfügung. Ein Objekt dieser Klasse ermöglicht es, einen *serialisierten* – d.h. auf Datenträger abgespeicherten – *Zähler* zu verwalten. Verschaffen Sie sich einen Überblick über die Funktionsweise der Klasse…

   Die privaten Methoden `loadCounter()` und `saveCounter()`, welche den Wert des Zählers von Datenträger lesen bzw. auf diesen schreiben, sind *synchronisiert* damit kein gleichzeitiges Lesen und/oder Schreiben des Zählerstandes von/auf Datenträger erfolgen kann.

   Erstellen Sie zuerst eine Klasse `CounterIncrementThread`, welche *tausendmal* den Zählerstand des Zählers inkrementiert. Dabei soll es möglich sein, Objekte dieser Klasse zu instanziieren, die parallel auf den Zähler zugreifen und das Inkrementieren erledigen.

   Erstellen Sie dann das Testprogramm `SerializedCounterTest`, welches zuerst den Zähler instanziieret und dann zwei Threads startet, welche *parallel* den Zähler inkrementieren. Dabei wird der Zähler nicht kontinuierlich erhöht. Wieso?

```
Old value: 0
New value: 1997
```

   Ändern Sie nun das Programm, so dass keine Nummern doppelt vergeben werden…

3. Ihnen werden die im Unterricht besprochenen Klassen, welche einen *Chatserver und -client* realisieren bereitgestellt. Sie sollen im Folgenden ein *Simulationsprogramm* schreiben, welches den gleichzeitigen Zugriff von mehreren Clients (z.B. 50) realisiert und das richtige Verhalten des Chatservers verifiziert.

   Nehmen Sie dazu als Vorlage die Klasse `ChatClient` her, und realisieren Sie ein einfaches Testprogramm mit dem Namen `ChatClientTest`. Dieses soll die Threads, welche die Chatclients simulieren, starten.

   Jeder Thread soll sich zuerst mit dem Server verbinden und dabei seinen Namen (z.B. den *Namen des Threads*) übermitteln. Dann soll er eine gewisse Anzahl von Malen (z.B. 50) – und jedes Mal etwas *zeitverzögert* – einen *beliebigen Text* zum Server schicken, bevor er nach einer Wartezeit von einer Sekunde sich beendet und somit die Verbindung zum Server wiederum abbricht (**HINWEIS:** Das Warten von einer Sekunde nach dem Senden der letzten Nachricht ist notwendig um zu verhindern, dass die Verbindung zum Server bereits abgebrochen wird, bevor dieser noch die Möglichkeit bekommen hat, die Nachricht des Clients entgegen zu nehmen).

4. Sollten Sie den Chatserver nicht richtig synchronisiert haben, so wird dieser *Fehlermeldungen* werfen bzw. am Ende der Simulation noch Objekte in der `ArrayList` `outputStreams` haben. Sorgen Sie deshalb dafür, dass auch Ihr Chatserver richtig synchronisiert ist.