Übung 11 - Bonusaufgabe: Threads

1. Monte-Carlo Simulation zur parallelen Schätzung von Pi

Punkte: 8

Lernziel: In dieser Aufgabe sollen Sie den Umgang mit *Threads* üben.

Mit Hilfe von Zufallszahlen ist es möglich, eine Schätzung der mathematischen Konstante π zu erhalten. Die Idee basiert auf der Tatsache, dass die Fläche eines Kreises mit Radius 1 gleich π ist ($A = r^2 \cdot \pi$), und die Fläche eines Viertels dieses Kreises daher gleich $\frac{\pi}{4}$ ist. Dieser Viertelkreis lässt sich in ein 1-mal-1 Quadrat mit Fläche 1 legen, wie in Abbildung 1 gezeigt.

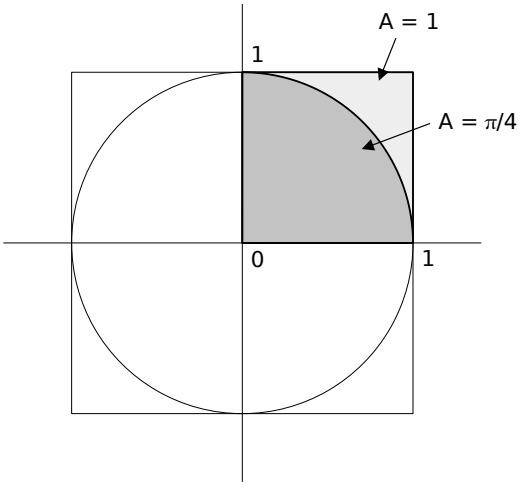

Abbildung 1: Viertelkreis mit Radius 1 innerhalb eines 1-mal-1 Quadrats.

Wird ein Punkt innerhalb dieses Quadrats zufällig gewählt, ist die Wahrscheinlichkeit, dass dieser Punkt innerhalb des Kreises liegt, $\frac{\pi}{4}$. Werden nun N Punkte zufällig gewählt, wobei C davon innerhalb des Kreises liegen, erwarten wir, dass der Anteil jener Punkte, die in den Kreis fallen, also $\frac{C}{N}$, etwa $\frac{\pi}{4}$ beträgt.[1] Wir erwarten also, dass $4 \cdot \frac{C}{N}$ ungefähr π beträgt. Wenn N groß ist, sollte dies eine ausreichend gute Schätzung für π sein.

Entwickeln Sie ein Java Programm, welches Anhand dieser Beschreibung die mathematische Konstante π schätzt. Eine Zufallszahl zwischen 0.0 und 1.0 kann mit der Methode `Math.random()` ermittelt werden. Ob ein Punkt (x, y) innerhalb des Viertelkreises liegt, kann mit Hilfe der Kreisgleichung ermittelt werden: Wenn $x * x + y * y < 1$ ist, liegt der Punkt innerhalb des Kreises. Um einen guten Schätzwert für π zu erhalten, müssen also viele Zufallszahlen generiert und Kreistests durchgeführt werden.

Gehen Sie dabei wie folgt vor:

- Erstellen Sie zwei Klassen `Stopper` und `PiMonteCarloSimulation`, die `Runnable` implementieren (Sie dürfen alternativ auch direkt von `Thread` erben).
- Klasse `Stopper`:
  - `Stopper` soll ein öffentliches statisches Feld `boolean stopped` enthalten, das anfangs auf `false` gesetzt ist.

[1] Diese Annahme ist nur richtig, wenn die Zufallszahlen gleichverteilt sind.

  - In der überschriebenen `run`-Methode soll, solange `stopped` noch auf `false` steht, eine Zufallszahl zwischen 0 und 1 generiert werden. Ist diese Zufallszahl ≥ 0.99 soll `stopped` auf `true` gesetzt werden, ansonsten soll sich der Thread für 100 Millisekunden schlafen legen und die Schleife anschließend erneut ausführen.

- Klasse `PiMonteCarloSimulation`:
  - `PiMonteCarloSimulation` soll private Felder für die Werte n (Anzahl der Versuche) und c (Anzahl der "Kreistreffer") enthalten. Initialisieren Sie beide mit 0 und stellen Sie passende Getter-Methoden zur Verfügung.
  - Implementieren Sie eine Methode `getPiEstimate`, welche den aktuellen Schätzwert von Pi ($4 \cdot \frac{c}{n}$) zurückliefert.
  - In der überschriebenen `run`-Methode soll, solange das statische Feld `Stopper.stopped` auf `false` steht,
    1. n um eins erhöht werden,
    2. ein zufälliges x und y generiert werden, und
    3. falls der Punkt (x, y) innerhalb des Kreises liegt, c um eins erhöht werden.
- Starten Sie in der `main`-Methode Ihres Programms nun 11 Threads: 10 für `PiMonteCarloSimulation` und einen für `Stopper`.
- Nach dem Starten aller Threads warten Sie mittels `join()` darauf, dass alle Threads ihre Arbeit abgeschlossen haben.
- Geben Sie dann abschließend den geschätzen π-Wert aller Simulationen aus. Summieren Sie außerdem alle C-Werte und N-Werte aller Threads auf und berechnen auf Basis dieser Summen ein "finales" Pi.

Beispielausgabe (Komma und Tausendertrennzeichen sind systemabhängig)

```
Starting 10 threads, each performing a monte carlo simulation ...
Starting Stopper thread ...
Waiting via .join() until Stopper thread generates a random number >= 0.99 ...
Stopper thread joined ...
Waiting via .join() for all simulation threads to finish ...
All simulation threads joined ...
Estimated pi of simulation #1: 3,142183 (4,0 * 2.370.571 / 3.017.738)
Estimated pi of simulation #2: 3,140656 (4,0 * 3.141.336 / 4.000.866)
Estimated pi of simulation #3: 3,140322 (4,0 * 3.037.957 / 3.869.612)
Estimated pi of simulation #4: 3,141236 (4,0 * 2.210.708 / 2.815.080)
Estimated pi of simulation #5: 3,142157 (4,0 * 2.619.155 / 3.334.213)
Estimated pi of simulation #6: 3,141651 (4,0 * 3.315.023 / 4.220.740)
Estimated pi of simulation #7: 3,142163 (4,0 * 2.838.148 / 3.612.986)
Estimated pi of simulation #8: 3,141559 (4,0 * 2.184.534 / 2.781.465)
Estimated pi of simulation #9: 3,141008 (4,0 * 2.417.725 / 3.078.916)
Estimated pi of simulation #10: 3,142041 (4,0 * 3.000.422 / 3.819.711)

Overall estimated pi: 3,141480 (4,0 * 27.135.579 / 34.551.327)
```

Zusätzliche Anweisungen

- Verwenden Sie zur Implementierung der Ein- und Ausgabe für alle Programme nur die Funktionen der beiden bereitgestellten Klassen `In` und `Out`, wie in der Übung gezeigt. Die entsprechenden Java-Dateien samt HTML-Dokumentation sind in Moodle unter *InOut.zip* zu finden.
- Implementieren Sie formatierte Ausgaben, indem Sie `String.format` in Kombination mit `Out.print` und `Out.println` verwenden, wie in der Übung gezeigt. Verwenden Sie als Referenz den Foliensatz `StringFormat.pdf` im Moodle.
- Sie dürfen, falls nötig, `Math`-Funktionen der Java-Bibliothek nutzen.
- Verwenden Sie für Variablen passende Datentypen Ihrer Wahl.
- Alle Namen (Klassennamen, Variablennamen, Methodennamen, etc.) sind in Englisch zu wählen.
- Formatierung, Namenswahl, etc. fließen in die Bewertung mit ein.
- Vermeiden Sie Codeduplikation.

Abzugeben

Geben Sie *eine .zip Datei* mit dem Namen kxxxxxxxx_UExx.zip ab (Beispiel: k01234567_UE11.zip). Darin muss enthalten sein:

- Der *Source Code* der aktuellen Übung (alle `.java` Dateien, keine `.class` Dateien).
- Ein Testprotokoll (`Testprotokoll.txt` oder `Testprotokoll.pdf`).