Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim Geisenheim

Fachbereich Design Informatik Medien
Studiengang Bachelor Allgemeine Informatik

Programmieren 3 AI – Praktikum

Übungsblatt 1

1 Punkt

Sebastian Flothow

2011-10-17

Abnahme: 2011-10-24

Zur Abnahme ist die persönliche Anwesenheit erforderlich. Bearbeiten Sie die Aufgaben so, dass sie nicht nur funktionierenden Code vorzuweisen haben, sondern diesen auch erläutern und Fragen dazu beantworten können. Zusätzlich sind die Lösungen am Tag der Abnahme per Email an `sebastian@flothow.de` einzusenden; dies dient u.a. der Prüfung auf eventuelle Plagiate.

Es muss für jede Aufgabe eine getrennte Lösung vorliegen, vorzugsweise in einem separaten Verzeichnis. Falls Aufgaben die Weiterverwendung der Lösung einer vorhergehenden Aufgabe vorsehen, fertigen Sie dazu Kopien der jeweiligen Quelltexte an. Wenn Sie mit Eclipse arbeiten, legen Sie für jede Aufgabe ein separates Projekt an, und verwenden Sie für die Email-Abgabe die Exportfunktion, um die Projekte als ZIP-Archive zu speichern.

Fakultät

In den Aufgaben dieses Blattes sollen Programme zur Berechnung der Fakultät verfasst werden. Für jede natürliche Zahl n ist ihre Fakultät $n!$ wie folgt definiert:

$$n! := \prod_{k=1}^{n} k$$

Die Fakultät kann auch rekursiv definiert werden, eine rekursive Implementation hätte jedoch für große n einen inakzeptablen Speicherbedarf. Verwenden Sie daher den iterativen Algorithmus, der aus obiger Definition natürlich folgt.

Beachten Sie, dass bereits 21! > `Long.MAX_VALUE`, die primitiven Ganzzahltypen sind also ungeeignet. Verwenden Sie stattdessen `java.math.BigInteger` für alle Berechnungen.

n	$n!$
0	1
1	1
2	2
3	6
4	24
5	120
17	355687428096000
23	25852016738884976640000
42	1405006117752879898543142606244511569936384000000000

Tabelle 1: *Einige Werte der Fakultät*

Aufgabe 1

Schreiben Sie eine Klasse `Factorial`. Implementieren Sie darin die Methode `public static BigInteger factorial(BigInteger n)`, die die Fakultät von `n` zurückgibt.

Schreiben Sie weiterhin die Klasse `Main` mit der üblichen Methode `public static void main(String[] args)`. Diese soll in einer Endlosschleife:

1. Auf der Konsole einen Prompt ausgeben, der den Benutzer zur Eingabe einer Zahl n auffordert
2. Eine Zeile von der Konsole lesen und, falls die Eingabe leer ist, die Schleife abbrechen
3. Die gelesene Eingabe zu einem `BigInteger` umwandeln
4. Vermittels der Methode `Factorial.factorial` $n!$ berechnen und ausgeben. Die Ausgabe soll dabei auch die eingegebene Zahl n enthalten (die Ausgabezeilen könnten also z.B. die Form „`5! = 120`“ haben).

Testen Sie das Programm mit den in Tabelle 1 gegebenen Werten, die Ergebnisse müssen übereinstimmen. Testen Sie danach auch deutlich größere n; ab etwa 10000 müsste die Dauer der Berechnung merklich ansteigen.

Aufgabe 2

Kopieren Sie die beiden Klassen von Aufgabe 1.

Erweitern Sie `Factorial` jetzt so, dass Objekte dieser Klasse als `Runnable` eine Fakultätsberechnung ausführen können. Versehen Sie die Klasse dazu mit einem Konstruktor,

der ein `BigInteger` entgegennimmt und in einem privaten Feld der Klasse speichert. Implementieren sie das Interface `Runnable`; die Methode `run` soll `factorial` mit dem gespeicherten Wert aufrufen und das Ergebnis auf der Konsole ausgeben. Verwenden Sie das gleiche Ausgabeformat wie bei Aufgabe 1.

Entfernen Sie aus der `main`-Methode den vierten Schritt des in Aufgabe 1 beschriebenen Ablaufs. Stattdessen soll ein neues `Factorial` mit dem gelesenen n erzeugt, einem neuen Thread übergeben und dieser gestartet werden.

Testen Sie das Programm: Es muss jetzt, auch bei großen Zahlen, nach jeder Eingabe sofort bereit für die nächste Eingabe sein. Bei zügiger Eingabe einer Folge von Werten unterschiedlicher Größenordnungen sollte zu beobachten sein, dass später gestartete Berechnungen bereits laufende „Überholen“ können. Wenn die Eingabeschleife durch Eingabe einer leeren Zeile verlassen wird und die `main`-Methode zurückkehrt, müssen noch laufende Berechnungen abgeschlossen werden, bevor das Programm sich beendet.

Aufgabe 3

Kopieren Sie die beiden Klassen von Aufgabe 1.

Diesmal soll `Factorial` von `Thread` erben und eine Eingabewarteschlange abarbeiten. Ergänzen Sie einen Konstruktor, dem eine `BlockingQueue<BigInteger>` übergeben wird. Überschreiben Sie die Methode `run` so, dass sie in einer Endlosschleife Elemente blockierend aus der Warteschlange entnimmt, ihre Fakultät berechnet und ausgibt. Verwenden Sie das gleiche Ausgabeformat wie in den vorhergehenden Aufgaben. Falls eine `InterruptedException` auftritt, soll die Schleife abbrechen und `run` zurückkehren.

Erzeugen Sie am Anfang der `main`-Methode eine `BlockingQueue<BigInteger>` (Hinweis: `BlockingQueue` ist ein Interface, wählen Sie von den dieses implementierenden Klassen eine geeignete aus). Erzeugen Sie ein `Factorial` mit dieser Warteschlange als Argument und starten Sie es. Entfernen Sie aus der Eingabeschleife den vierten Schritt des in Aufgabe 1 beschriebenen Ablaufs. Stattdessen soll der gelesene Wert blockierend in die Warteschlange eingereiht werden. Nach Beendigung der Eingabeschleife soll der `Factorial`-Thread unterbrochen werden; es soll also nur noch die laufende Berechnung zuendegebracht, aber kein weiterer Wert mehr aus der Warteschlange entnommen werden.

Testen Sie wieder mit einer abwechslungsreichen Eingabefolge.

Aufgabe 4

Kopieren Sie die beiden Klassen von Aufgabe 2.

Ändern Sie die Klasse `Factorial` so, dass sie anstelle von `Runnable` das Interface `Callable<BigInteger>` implementiert, benennen Sie also auch die Methode `run` zu `call` um und passen Sie ihre Signatur an. Die Methode soll außerdem keine Ausgabe mehr machen, sondern das Ergebnis der Berechnung zurückgeben.

Die `main`-Methode soll zu Beginn einen `ExecutorService` erzeugen, der mit einem Thread-Pool fester Größe arbeitet. Verwenden Sie hierzu die Hilfsklasse `Executors`. Die

Anzahl der Threads soll der Anzahl der verfügbaren Prozessoren entsprechen. In der Eingabeschleife soll wie bei Aufgabe 2 im letzten Schritt ein neues `Factorial`-Objekt erzeugt werden. Anstatt jedoch für jede Berechnung einen eigenen Thread zu erzeugen, soll das `Factorial`-Objekt beim `ExecutorService` zur Ausführung eingereicht werden. Das zurückgegebene `Future`-Objekt soll in einer Liste gespeichert werden. Nach dem Ende der Eingabeschleife soll `main` über die gespeicherten `Future`s iterieren und die Ergebnisse ausgeben.