Bearbeitungszeitraum: eine Woche

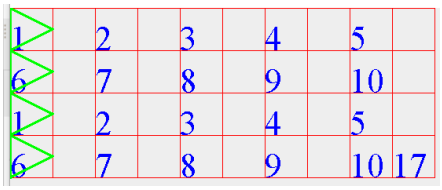
Die folgenden Aufgaben sollen den Gebrauch einiger Synchronisationsklassen aus dem java.util.concurrent Paket einüben. Sie lehnen sich an die Aufgaben aus dem Buch „Parallele Programmierung spielend gelernt mit dem Java-Hamster-Modell“ von Dietrich Boles an.

Es geht darum, dass Sucher auf einem rechteckigen Territorium aus Kacheln mit Körnern arbeiten. Auf jeder Kachel können sich eine bestimmte Anzahl von Körnern befinden. Jeder Sucher ist ein Thread. Sie sollen verschiedene Aufgaben mit unterschiedlich konfigurierten Territorien lösen. In einer Aufgabe sollen z.B. in jeder Zeile des Territoriums jeweils ein Sucher alle Körner finden.

Rahmenwerk für die Sucher Aufgaben

Für alle Aufgaben zusammen gibt es schon Code, den Sie verwenden müssen. Er ist zusammen mit dieser Aufgabenstellung auf Felix zu finden. Alle Klassen befinden sich in einem package sucher. Sie sollen den Code so vervollständigen, dass er die im Folgenden dargestellten Aufgaben erfüllt.

Territorium



In der obigen Abbildung sieht man ein Beispiel für ein Territorium. Es besteht aus rot umrandeten Kacheln. Sie sind in Zeilen und Spalten angeordnet. Auf manchen Kacheln befindet sich eine bestimmte Anzahl von Körnern, die hier durch eine blaue Zahl angegeben ist. Auf Kacheln ohne blaue Zahl befinden sich keine Körner. In der Spalte am linken Rand (Spaltenindex 0) befindet sich in diesem Beispiel auf jeder Kachel ein grünes Dreieck. Dies symbolisiert einen Sucher, der die Kacheln bearbeitet und z.B. die Körnerzahl auf einer Kachel feststellen soll. Die Spitze jedes Dreiecks zeigt hier nach rechts. Dies ist die Richtung, in die die Sucher sich bewegen. Die Sucher sollen sich Kachel für Kachel nach rechts bewegen, und dabei z.B. die gefundenen Körnerzahlen zusammenzählen.

Grundschleife

Für jede Aufgabe gibt es im Rahmenwerk eine Klasse. Diese Klassen sind von SucherImpl abgeleitet.
Jede Klasse hat eine Methode, die von einem Thread ausgeführt wird, z.B. run, compute oder call. Alle diese Methoden enthalten die gleiche Grundschleife:

```
boolean istKachel = true;

do {
     bearbeiteKachel();
     istKachel = bewege();
} while (istKachel);
```

Der Sucher hat eine aktuelle Kachel, die er gerade bearbeitet. Die Position dieser Kachel ist in dem Attribut `aktuellePosition` gespeichert. Die Methode `bearbeiteKachel()` führt für die aktuelle Kachel das aus, was mit der Kachel gemacht werden soll z.B. die Körnerzahl der Kachel zur Gesamtzahl der Körner hinzuaddieren. Die Methode `bearbeiteKachel()` muss von Ihnen je nach Aufgabe implementiert werden.
Die Methode `bewege()` bewegt die aktuelle Position des Suchers eine Kachel weiter. Die Methode `bewege()` wird geerbt und muss nicht implementiert werden. Die Bewegung geschieht in die Richtung, die das Attribut `gehRichtung` von `SucherImpl` vorgibt. Wenn die Bewegung über den Rand des Territoriums hinausführen würde, wird die aktuelle Position nicht verändert. Die Methode `bewege` liefert dann false. Die Grundschleife ist dann beendet.

Wie werden Zeilenindex und Spaltenindex einer Kachel gespeichert?

Der Zeilenindex und der Spaltenindex befindet sich in dem Attribut `aktuellePosition` von `SucherImpl`. Es ist vom Typ `int` Array (`int[]`). Der erste Arrayplatz ist der Zeilenindex und der zweite Arrayplatz ist der Spaltenindex. Alle Positionen sollen als solche `int` Arrays mit zwei Arrayplätzen auftreten.

Zentrale Synchronisationsobjekte

Immer wieder werden in den Aufgaben zentrale Objekt benötigt, mit deren Hilfe sich die Threads synchronisieren oder sogar Daten miteinander austauschen. Dabei ist wichtig, dass es von diesen zentralen Objekten immer nur ein Exemplar gibt und die Threads auf dasselbe Exemplar zugreifen. So gibt es z.B. die Tauscher Aufgabe, in der zwei Threads mit Hilfe eines `Exchanger` Objekts Zahlen austauschen. Dort hilft es nichts, wenn jeder der beiden Threads sein eigenes `Exchanger` Objekt hat, sondern beide Threads müssen auf dasselbe `Exchanger` Objekt zugreifen. In solchen Fällen wird in dem Konstruktor der Aufgabenklasse ein zusätzlicher Parameter für das Synchronisationsobjekt vorgesehen. Im Konstruktor wird das Objekt in einem Attribut gespeichert. Dieses Attribut kann dann z.B. in der `run` Methode benutzt werden. In der `main` Methode wird das Synchronisationsobjekt einmal erzeugt und den Konstruktoren der Aufgabenobjekte übergeben.

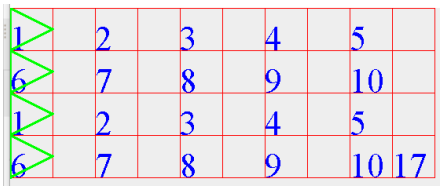
Sucher / SucherImpl

Die Methode `gibPosition` von `SucherImpl` liefert die Position der aktuellen Kachel. Die Methode `gibTerritorium` von `SucherImpl` liefert das Territorium.

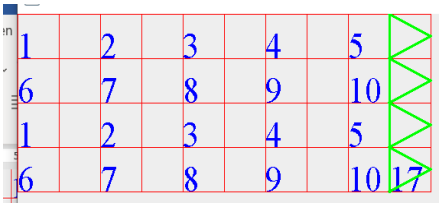
1 Aufgabe: Körner zählen

1.1 Beschreibung der Aufgabe

vorher



nachher



In der obigen Abbildung sieht man das Territorium. Es besteht aus 40 rot umrandeten Kacheln. Sie sind in 4 Zeilen und 10 Spalten angeordnet. Die Sucher (grüne Dreiecke) sollen sich Kachel für Kachel nach rechts bewegen, und dabei die gefundenen Körnerzahlen zusammenzählen. Zum Schluss sollen die Ergebnisse aller 4 Sucher aufaddiert und das Ergebnis ausgegeben werden:

```
Die Anzahl der Koerner betraegt: 127
```

Am Ende befinden sich alle Sucher in der Spalte am rechten Rand.

1.2 Verwendete Klassen / Interfaces von java.util.concurrent

1.2.1 Callable

Das Interface `Callable` ist der Zwilling zu `Runnable`. Man kann es einem Threadpool zum Ausführen mit der Methode `submit` übergeben, so wie man ein `Runnable` einem Thread Objekt zur Ausführung übergeben kann.
Nach dem Aufruf von `submit` führt ein Thread des Threadpools die Methode `call` aus, die das Interface `Callable` seinen Objekten vorschreibt. `call` ist der Zwilling zu `run` von `Runnable`. Aber im Unterschied zu `run` gibt die Methode `call` etwas zurück – in dieser Aufgabe einen Integer Wert.

1.2.2 ExecutorService / Threadpool

Ein Threadpool ist eine Menge von Threads, die `Runnable` oder `Callable` Objekte ausführen. Man kann sich mit der sogenannten Fabrikmethode

```
ExecutorService pool =
Executors.newFixedThreadPool(ZAHL_GLEICHZEITIGE_THREADS)
```

ein solches Threadpool Objekt vom Typ `ExecutorService` beschaffen. Den Threads im Threadpool kann man mit `submit(Callable)` die Callables zum Ausführen übergeben.
Im Übergabeparameter der Methode `newFixedThreadPool` kann man bestimmen, wieviel Threads im Pool auf `Callable` Objekte zum Ausführen warten sollen. Die Threads werden nach Ausführung von `Callable` Objekten nicht zerstört. Sie können daher für weitere `Callable` Objekte wiederverwendet werden. Dadurch wird die Zeit zur Erzeugung von neuen Thread Objekten gespart. Selbst wenn alle Threads im Pool schon mit `Callable` Objekten beschäftigt sind, kann man den Threads trotzdem immer neue Aufträge erteilen, indem man weiterhin die Methode `submit(Callable)` aufruft. Das neu übergebene `Callable` wird dann in einer Queue gespeichert und später von den Threads ausgeführt.

Es ist sinnvoll, in einem Threadpool nur so viele Threads zu halten, wie Ihr Computer Kerne hat. Eine Ausnahme bildet die Körner Esser Aufgabe.

1.2.3 Future

Wie kommen wir an den Rückgabewert heran, der von der `call` Methode eines `Callable` Objekts zurückgegeben wird, wenn es vom Threadpool ausgeführt worden ist?
Dazu dient ein `Future` Objekt, welches wir beim Aufruf von `submit` zurückgeliefert bekommen. Dieses `Future` Objekt hat eine `get` Methode, die das Ergebnis der Methode `call` liefert. Wenn wir z.B. mit dem main Thread `get` aufrufen, wartet der main Thread in der `get` Methode so lange, bis `call` ein Ergebnis liefert.

1.2.4 Beispiel

```
ExecutorService pool =
                Executors.newFixedThreadPool(4);
Callable<Integer> callable = new Callable<>() {

    public Integer call() throws Exception {
        return 2 + 2;
    }

};
Future<Integer> future = pool.submit(callable);
System.out.println(future.get());
```

1.3 Durchführung

1.3.1 Die Klasse KoernerZaehler

Sie sollen die Klassen des Pakets `sucher` verwenden. Dort befindet sich eine Klasse mit dem Namen `KoernerZaehler`. Die Klasse `KoernerZaehler` ist eine Unterklasse von `SucherImpl`. Sie implementiert außerdem das Interface `Callable<Integer>`.
Die Klasse `KoernerZaehler` ist unvollständig. Die Stellen, die fehlen, sind in der Klasse als Kommentare beschrieben. Ihre Aufgabe besteht daher darin, die Kommentare in Code zu verwandeln. Das, was programmiert werden soll, ist in den Kommentaren beschrieben.

1.3.2 Methoden, die vervollständigt werden müssen

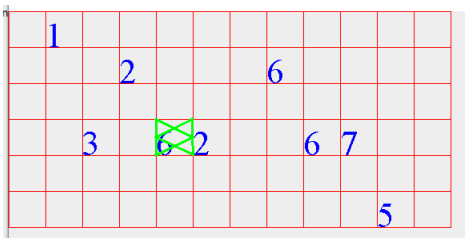
Was implementiert werden soll ist im Kommentar in den Methoden beschrieben.

- `bearbeiteKachel`
- `main`

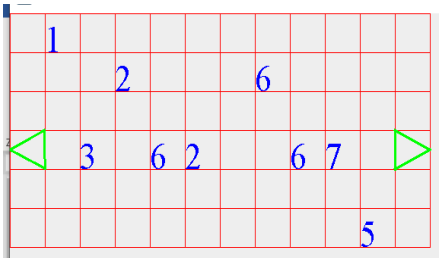
2 Aufgabe: Größte Körnerzahl in einer Zeile ermitteln

2.1 Beschreibung der Aufgabe

vorher



nachher



2.1.1 Was ist zu tun?

In einer bestimmten Zeile ist die größte Körnerzahl herauszufinden. In der obigen Abbildung sieht man das Territorium für diese Aufgabe. Die Zeilenindizes beginnen mit 0. In der Zeile 3 ist die größte Körnerzahl 7 und in der Zeile 1 ist die größte Körnerzahl 6. Wir wollen die größte Körnerzahl für die Zeile 3 mit zwei Suchern ermitteln.

2.1.2 Wie ist es zu tun?

Die zwei Sucher (grüne Dreiecke) starten beide auf der Kachel mit Zeilenindex 3 und Spaltenindex 4. Ein Sucher soll sich Kachel für Kachel nach rechts bewegen, und dabei die größte Körnerzahl auf den Kacheln herausfinden. Der andere Sucher bewegt sich gleichzeitig nach links und sucht dort nach der größten Körnerzahl. Zum Schluss werden die Ergebnisse der beiden Sucher verglichen und das größere ausgegeben:

```
Die maximale Anzahl an Koernern auf Kacheln in
der Reihe, in der ich stehe, betraegt: 7
```

Am Ende befindet sich der eine Sucher am linken Rand und der andere Sucher am rechten Rand.

2.2 Verwendete Klassen / Interfaces von java.util.concurrent

2.2.1 CountDownLatch

Ein `CountDownLatch` ist ein Objekt, das einen Integer Wert hat. Dieser Wert wird zunächst mit dem Konstruktor festgelegt. Mit der Methode `countDown`() können Threads diesen Wert jeweils um eins erniedrigen. Dies geschieht so lange, bis der Wert null beträgt. Wenn Threads die Methode `await`() aufrufen, bleiben sie so lange in der Methode gefangen, bis der Wert auf null abgesenkt wurde. Dann kommen alle gefangenen Threads gleichzeitig aus der `await`() Methode wieder heraus. Ein `CountDownLatch`, dessen Wert auf null abgesunken ist, kann nicht mehr zurückgesetzt werden.

2.3 Durchführung

2.3.1 Die Klasse KoernerMaximal

Sie sollen die Klassen des Pakets `sucher` verwenden. Dort befindet sich eine Klasse mit dem Namen `KoernerMaximal`. Die Klasse `KoernerMaximal` ist eine Unterklasse von `SucherImpl`

und daher auch eine Unterklasse von `Thread`. Ein `KoernerMaximal` Objekt kann also mit `start`() als Thread gestartet werden und `run`() wird dann ausgeführt.
Die Klasse `KoernerMaximal` ist unvollständig. Die Stellen, die fehlen, sind in der Klasse als Kommentare beschrieben. Ihre Aufgabe besteht daher darin, die Kommentare in Code zu verwandeln. Das, was programmiert werden soll, ist in den Kommentaren beschrieben.

2.3.2 Methoden, die vervollständigt werden müssen

- `run`
- `main`
- `bearbeiteKachel`
- Konstruktor von `KoernerMaximal`

2.3.3 Die Methode run

Weil `KoernerZaehler` ein Thread ist, muss die `run` Methode implementiert sein. Die oben beschriebene Grundschleife ist bereits in der Methode `run` enthalten. In `run` muss noch etwas in Zusammenhang mit der Synchronisation mit dem `main` Thread programmiert werden. Die Suche nach dem Maximum der Körnerzahlen auf der Kachel der aktuellen Position findet in der Methode `bearbeiteKachel`() statt.

2.3.4 Die Methode main

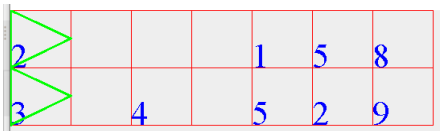
Was implementiert werden soll ist im Kommentar beschrieben. Sie sollen in dieser Aufgabe mit Hilfe eines `CountDownLatch` Objekts den `main` Thread so lange stoppen, bis die beiden Helfer Threads ihre Arbeit beendet und jeweils für sich die maximal Körnerzahl auf ihren Kacheln bestimmt haben.

3 Aufgabe: Körnerzahlen von oberer und unterer Zeile tauschen

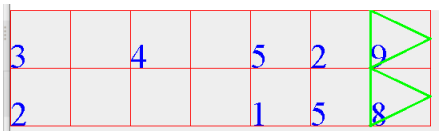
3.1 Beschreibung der Aufgabe

3.1.1 Was ist zu tun?

voher



nachher



Es geht darum die Inhalte zweier Zeilen zu tauschen. In der obigen Abbildung sieht man das Territorium für diese Aufgabe. Es besteht aus zwei Zeilen. Dort steht z.B. in der unteren Zeile der Spalte mit dem Index 0 die Körnerzahl 3 und in der oberen Zeile derselben Spalte die Körnerzahl 2. Nach dem Tausch soll oben die 2 und unten die 3 stehen. Dies soll für alle Spalten durchgeführt werden.

3.1.2 Wie ist es zu tun?

Zwei Sucher werden gestartet. Sie durchlaufen jeweils eine Zeile von links nach rechts. Dabei warten sie bei jeder Spalte aufeinander und teilen sich dann gegenseitig die Körnerzahlen ihrer Kacheln mit. Dann erneuern sie die Körnerzahl ihrer Kachel mit dem mitgeteilten Wert und gehen weiter.

Zum Schluss werden die Körnerzahlen des Territoriums ausgegeben:

```
3    0    4    0    5    2    9
2    0    0    0    1    5    8
```

Am Ende befinden sich beide Sucher am rechten Rand.

3.2 Verwendete Klassen / Interfaces von java.util.concurrent

3.2.1 Threadpool / ExecutorService

Siehe 1.2.2. An dieser Aufgabe ist zu erkennen, dass man bei einem `ExecutorService` den dort vorhandenen Threads auch `Runnable` Objekte (und nicht nur `Callable` Objekte) zum Abarbeiten übergeben kann. Dies geschieht mit der Methode `execute(Runnable)`. Werden dem `ExecutorService` mehr `Runnable` Objekte übergeben, als gerade abgearbeitet werden können, dann werden die überschüssigen `Runnable` Objekte in einer Queue gespeichert. Sobald die Threads im `ExecutorService` mit einem `Runnable` fertig sind, holen sie das nächste `Runnable` aus der Queue und bearbeiten es.

3.2.2 Exchanger

Mit einem `Exchanger` Objekt können zwei Threads zwei Werte austauschen. Wenn ein Thread A die `exchange(Integer)` Methode eines `Exchanger<Integer>` Objekts aufruft, wird er so lange in der Methode festgehalten, bis ein weiterer Thread B ebenfalls die `exchange(Integer)` Methode desselben Exchanger Objekts aufruft. Angenommen Thread A hat der `exchange` Methode den Wert 2 übergeben und Thread B hat der `exchange` Methode den Wert 3 übergeben. Dann erhält Thread A den Wert 3 von Thread B als Rückgabewert der `exchange` Methode. Umgekehrt erhält Thread B den Wert 2von Thread A als Rückgabewert. Damit haben die Threads die Werte ausgetauscht und können der `exchange` Methode beide entkommen.

3.3 Durchführung

3.3.1 Die Klasse KoernerTausch

Sie sollen die Klassen des Pakets `sucher` verwenden. Dort befindet sich eine Klasse mit dem Namen `KoernerTausch`. Diese Klasse ist eine Unterklasse von `SucherImpl` und daher auch eine Unterklasse von `Thread` und `Runnable`. Ein `KoernerTausch` Objekt kann also einem `ExecutorService` zur Ausführung übergeben werden.
Die Klasse `KoernerTausch` ist unvollständig. Die Stellen, die fehlen, sind in der Klasse als Kommentare beschrieben. Ihre Aufgabe besteht daher darin, die Kommentare in Code zu verwandeln. Das, was programmiert werden soll, ist in den Kommentaren beschrieben.

3.3.2 Methoden, die vervollständigt werden müssen

- `main`
- `bearbeiteKachel`