Aufgabe V5: Kritische Abschnitte

Die Java–Klasse drei Seiten weiter implementiert einen Puffer mittels eines Arrays und eines Zählers. Der Puffer arbeitet nach dem LIFO–Prinzip (engl.: *Last In, First Out*), wie ein Stapel. Er steht stellvertretend für beliebige, nicht–triviale Datenstrukturen. Die Implementierung ist nicht gegen gleichzeitige Aufrufe durch mehrere Prozesse abgesichert.
Im Folgenden rufen zwei Prozesse, d.h. Java–Threads, gleichzeitig Methoden am selben Puffer auf. Der Puffer enthält bereits einige Werte, aber auch noch freie Plätze. Die Operationen stoßen also weder an den oberen noch an den unteren Rand des Arrays. Gehen Sie davon aus, dass zwischen zwei Zeilen des Quelltexts jeweils der andere Prozess zum Zuge kommen könnte.
Wenn beide Prozesse `fetch` aufrufen wäre das korrekte Verhalten, dass einer den obersten Wert vom Stapel nimmt, der andere den zweitobersten. Das kann aber schiefgehen. Bei folgender Reihenfolge der Befehle entnehmen beide Prozesse den obersten Wert:

Prozess A: `fetch`	**Prozess B**: `fetch`
which_A = items-1	
	which_B = items-1
result_A = buffer[which_A]	
	result_B = buffer[which_B]
items = which_A	
	items = which_B
return result_A	
	return result_B

Lokale Variablen wie `which` und `result` existieren pro Prozess, deshalb die Indizes $_A$ und $_B$. Im Quelltext sind sie als `final` deklariert. Die Attribute `items` und `buffer` des Puffers sind für beide Prozesse die selben.

Lösung ab der nächsten Seite...

a) Welche Zeilen muss Prozess A am Stück ausführen, damit Prozess B nicht mehr den gleichen Wert vom Stapel nehmen kann?

Diese Anweisungen bilden einen *kritischen Abschnitt* in `fetch`.

Prozess A muss die ersten drei Anweisungen am Stück ausführen, vom Lesen des gemeinsamen Attributs `items` bis zum Schreiben mit der neuen Anzahl.

Prozess A: `fetch`	**Prozess B**: `fetch`
which_A = **items**-1	
result_A = buffer[which_A]	
items = which_A	
	which_B = **items**-1
	result_B = buffer[which_B]
	items = which_B
return result_A	
	return result_B

Das gilt auch für Prozess B, falls dieser vor A auf den Puffer zugreift, oder wenn noch ein dritter Prozess C ins Spiel kommt. Kritische Abschnitte müssen immer am Stück ausgeführt werden, um die jeweilige Datenstruktur wieder in einen konsistenten Zustand zu bringen.

b) Beide Prozesse rufen `store` auf. Was ist das korrekte Verhalten? Was könnte wie schiefgehen? Welche Anweisungen bilden den kritischen Abschnitt in `store`?

Die Werte beider Prozesse sollten nacheinander im Puffer landen. Aber wenn beide Prozesse die gleiche Anzahl aus `items` lesen, wird einer den abgelegten Wert des anderen überschreiben. Das könnte so aussehen:

Prozess A: `store`	**Prozess B**: `store`
which_A = items	
	which_B = items
buffer[which_A] = item_A	
	buffer[which_B] = item_B
items = which_A+1	
	items = which_B+1

Der kritische Abschnitt umfasst alle drei Zeilen von `store`, wiederum vom Lesen des gemeinsamen Attributs `items` bis zum Schreiben mit der neuen Anzahl.

c) Prozess A ruft `fetch`, Prozess B `store`. Welches sind die zwei korrekten Verhalten bei dieser Kombination? Auf welche zwei Arten könnte es schiefgehen? Gibt es neue kritische Abschnitte für die Kombination?

Bei korrektem Verhalten legt entweder Prozess A einen Wert ab und B nimmt diesen, oder Prozess B nimmt einen Wert vom Stapel, bevor Prozess A seinen dort ablegt.

Im einen Fehlerfall bleibt der von A gelesene Wert auf dem Stapel:

Prozess A: `fetch`	**Prozess B**: `store`
which_A = items-1	
	which_B = items
result = buffer[which_A]	
	buffer[which_B] = item
items = which_A	
	items = which_B+1

Im anderen Fehlerfall geht der von `store` abgelegte Wert verloren.

Prozess A: `fetch`	**Prozess B**: `store`
	which_B = items
which_A = items-1	
	buffer[which_B] = item
result = buffer[which_A]	
	items = which_B+1
items = which_A	

Es gibt keine neuen kritischen Abschnitte. Wenn die kritischen Abschnitte aus den ersten beiden Teilaufgaben am Stück ablaufen, kann keiner der beiden Fehlerfälle auftreten.

d) Prozess A ruft `store`, Prozess B `toString`. Welches sind die zwei korrekten Verhalten bei dieser Kombination? Was könnte wie schiefgehen? Welche Anweisungen bilden den kritischen Abschnitt in `toString`?

`toString` muss entweder den alten oder den neuen Wert von `items` ausgeben, beides ist korrekt. Da `toString` nur einmal auf `items` zugreift, und weder `store` noch `fetch` ungültige Zwischenwerte in `items` schreiben, geht hier nichts schief. Fehler könnten entstehen, wenn `toString` auch die Inhalte des Arrays ausgeben würde. Dazu müssten mehrere Lesezugriffe erfolgen, die gemeinsam einen kritischen Abschnitt bilden.

Im Code der Aufgabe besteht der kritische Abschnitt in `toString` nur aus dem einen Zugriff auf `items`. Das Speichermodell von Java garantiert für den Datentyp `int`, dass jeder Zugriff atomar erfolgt. Für `long` ist das nicht der Fall, ein solches Attribut könnte in zwei Hälften gelesen werden, die Teile des alten und des neuen Werts ungültig kombinieren.

```
import java.lang.reflect.Array;


public class BufferLIFO<T>
{
  protected final T[] buffer;
  protected       int items;

  @SuppressWarnings("unchecked")
  public BufferLIFO(Class<T> clazz, int capacity)
   {
     buffer = // new T[capacity]
       (T[]) Array.newInstance(clazz, capacity);
     items  = 0;
   }

  public void store(T item)
   {
     final int which = items;
     buffer[which] = item;
     items = which+1;
   }

  public T fetch()
   {
     final int which = items-1;
     final T result = buffer[which];
     items = which;
     return result;
   }

  public String toString()
   {
     StringBuilder sb = new StringBuilder(80);
     sb.append(getClass().getName())
       .append('[').append(items)
       .append('/').append(buffer.length)
       .append(']');
     return sb.toString();
   }
}
```