Laboratorium Nr 7
"Wzorce projektowe dla programowania współbieżnego"
Radosław Kopeć
17.11.2020

I. Zadanie 1

1. Treść zadania

Zaimplementować bufor jako aktywny obiekt
(Producenci-Konsumenci)
Wskazówki:

Pracownik powinien implementować samą kolejkę (bufor) oraz dodatkowe metody (czyPusty etc.), które pomogą w implementacji strażników. W klasie tej powinna być tylko funkcjonalność, ale nie logika związana z synchronizacją.

Dla każdej metody aktywnego obiektu powinna być specjalizacja klasy MethodRequest. W tej klasie m.in. zaimplementowana jest metoda guard(), która oblicza spełnienie warunków synchronizacji (korzystając z metod dostarczonych przez Pracownika).

Proxy wykonuje się w wątku klienta, który wywołuje metodę. Tworzenie Method request i kolejkowanie jej w Activation queue odbywa się również w wątku klienta. Servant i Scheduler wykonują się w osobnym (oba w tym samym) wątku.

2. Koncepcja rozwiązania

Bufor nie będzie wiedział nic o synchronizacji dostępu do niego samego. Synchronizacja została wyekstrahowana do funkcji Schedulera i funkcji guard(). Scheduler zarządza kolejką operacji bufora, pozwala dodawać metody, oraz sam wybiera operacje do przeprowadzenia na buforze. Proxy zapewnia dostęp do Bufora poprzez scheduler, dodaje odpowiednie metody do kolejki. Metody są reprezentowane przez klasę abstrakcyjną, a ich wynik jest powiązany z obiektem FutureResult. Producent i Konsument będą zamiast bufora używać ProxyBufora. Klasy FutureResult, oraz BufferMethodRequest jako klasy generyczne, muszą one uwzględniać typ zwracanego wyniku.

3. **Implementacja i wyniki:**

Bufor (Servant):

```
public class Buffer {

    private final int[] buffer;
    private final int size;
    private final boolean[] canWrite;

    public Buffer(int size){
        this.size = size;
        buffer = new int[size];
        canWrite =  new boolean[size];
        for (int i =0;i < size; i++){
            buffer[i]= 0;
            canWrite[i] = true;
        }
    }

    public boolean canWrite(){

        for (int i = 0; i< size; i++) {
            if(canWrite[i]){
                return true;
            }
        }
        return false;
    }

    public boolean canRead(){
        for (int i = 0; i< size; i++) {
            if(!canWrite[i]){
                return true;
            }
        }
        return false;
    }
```

```java
private int findFieldToWrite(){
    for(int i=0;i<size;i++){
        if(canWrite[i]){
            return i;
        }
    }
    throw new IllegalStateException("at least one files should be free");

}

private int findFieldToRead(){
    for(int i=0;i<size;i++){
        if(!canWrite[i]){
            return i;
        }
    }
    throw new IllegalStateException("at least one files should be free");
}
```

```java
public void put(int value) {

    int index = findFieldToWrite();
    buffer[index] = value;
    canWrite[index] = false;

}


public int get() {
    int index = findFieldToRead();
    canWrite[index] = true;
    return buffer[index];
}
```

BufferMethodRequest:

```java
public abstract class BufferMethodRequest<R> {

    Buffer buffer;
    FutureResult<R> future;

    public BufferMethodRequest(Buffer buffer,FutureResult<R> future) {
        this.buffer = buffer;
        this.future = future;
    }

    abstract boolean guard();
    abstract void call();

}
```

BufferGetRequest:

```java
public class BufferGetRequest extends BufferMethodRequest<Integer> {

    public BufferGetRequest(Buffer buffer, FutureResult<Integer> future) {
        super(buffer, future);
    }

    @Override
    public boolean guard() {
        return buffer.canRead();
    }

    @Override
    public void call() {
        int result =  buffer.get();
        future.setResult(result);
        future.resolve();
    }
}
```

BufferPutRequest:

```java
public class BufferPutRequest extends BufferMethodRequest<Void>{

    private final int value;

    public BufferPutRequest(Buffer buffer, FutureResult<Void> future, int value) {
        super(buffer, future);
        this.value = value;
    }

    @Override
    public boolean guard() { return buffer.canWrite(); }

    @Override
    public void call() {
        buffer.put(value);
        future.resolve();
    }

}
```

BufferProxy:

```java
public class BufferProxy {

    BufferScheduler scheduler;
    Buffer buffer;

    public BufferProxy(int size) {
        buffer = new Buffer(size);
        scheduler = new BufferScheduler(buffer);
        scheduler.start();
    }

    FutureResult<Integer> get(){
        FutureResult<Integer> result = new FutureResult<>();
        BufferGetRequest method = new BufferGetRequest(buffer,result);
        scheduler.enqueue(method);
        return result;
    }

    FutureResult<Void> put(int value){
        FutureResult<Void> result = new FutureResult<>();
        BufferPutRequest method = new BufferPutRequest(buffer,result,value);
        scheduler.enqueue(method);
        return result;
    }
}
```

BufferScheduler:

```java
public class BufferScheduler extends Thread {

    LinkedBlockingQueue<BufferMethodRequest<?>> activeQueue = new LinkedBlockingQueue<>();
    Buffer buffer;

    public BufferScheduler(Buffer buffer) {
        this.buffer = buffer;
    }


    public void enqueue(BufferMethodRequest<?> method){
        try {
            activeQueue.put(method);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
```

```java
    public void dispatch()   {

        try {
            BufferMethodRequest<?> method = activeQueue.take();

                if (method.guard()) {
                    method.call();
                    activeQueue.remove(method);
                    System.out.println(buffer);
                }
            else{
                activeQueue.put(method);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run(){
        System.out.println("Run Scheduler");
        while(true){
            dispatch();
        }
    }
}
```

Consumer:

```
class Consumer extends Thread {
    private BufferProxy _buf;

    public Consumer(BufferProxy buffer){
        super();
        this._buf = buffer;
    }

    public void run() {
      for (int i = 0; i < 100; ++i) {
          System.out.println("I get the: " + _buf.get().get());
      }
    }
}
```

Prodcuer:

```
class Producer extends Thread {
    private BufferProxy _buf;

    public Producer(BufferProxy buffer){
        super();
        this._buf = buffer;
    }

    public void run() {

        for (int i = 0; i < 100; ++i) {
            _buf.put(i);
            System.out.println("I add the " + i);
        }
    }
}
```

Testy:

```
public class PodpunktA {
    public static void main(String[] args) {
        BufferProxy buffer = new BufferProxy( size: 10);

        Producer producer = new Producer(buffer);
        Consumer consumer = new Consumer(buffer);

        producer.start();
        consumer.start();
    }
}
```

```
public class PodpunktB {
    public static void main(String[] args) {
        BufferProxy buffer = new BufferProxy( size: 10);
        int n1 = 5;
        int n2 = 5;

        for(int i=0;i<n1 ;i++){
            Producer producer = new Producer(buffer);
            producer.start();
        }
        for(int i=0;i<n2 ;i++){
            Consumer consumer = new Consumer(buffer);
            consumer.start();
        }

    }
}
```

4. Wnioski

Zalety Active Object:

- Wzmacnia współbieżność w aplikacji równocześnie redukując skomplikowanie operacji asynchronicznych. Szczególnie jeśli obiekty komunikują się poprzez wiadomości message-driven applications.
- Wzorzec Active Object wykorzystuje współbieżność oferowaną przez platformę sprzętową.

Wady active object:

- Może nastąpić sytuacja, że dla wielu wiadomości nastąpi zapchanie kolejki, lub narzut operacji związanych z zarządzaniem kolejną, zmianą kontekstu, przekazywaniem danych będzie tak duży, że nie będzie opłacało się implementować Active Object.

- Debugowanie tego typu rozwiązań jest trudne.

5. Bibliografia

https://www.dre.vanderbilt.edu/~schmidt/PDF/Active-Objects.pdf
https://pl.wikipedia.org/wiki/Active_object
https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html
https://www.baeldung.com/java-blocking-queue
https://www.youtube.com/watch?v=aZ2xxVxhbBU

Radosław Kopeć