Lab 07

Wzorce projektowe dla programowania współbieżnego

Adrian Madej 19.11.2023

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. Rozwiązanie zadania

Koncepcja

Problem zostanie rozwiązany w oparciu o wzorzec Active Object. Aktywny obiekt oddziela wykonanie metody od wywołania metody aby poprawić współbieżność i uprościć synchronizowany dostęp do obiektów, które są umieszczone w swoich własnych wątkach; dlatego też, wywołanie metody odbywa się w wątku klienta, a jej wykonanie w wątku pracownika.

Implementacja

Klasa Servant działa jako prosty kontener. Metody put i get pozwalają na dodawanie i pobieranie elementów z kolejki. Implementacja nie zawiera dodatkowej logiki zabezpieczającej dostęp do współdzielonego zasobu w środowisku wielowątkowym, lecz samą funkcjonalność.

```
public class Servant {
    private final Queue<Integer> queue = new LinkedList<>();

    private final int capacity;

    public Servant(int capacity) {
        this.capacity = capacity;
    }

    public void put(List<Integer> data){
        queue.addAll(data);
    }

    public List<Integer> get(int elements){
        List<Integer> out = new LinkedList<>();
        while(elements-- > 0){
            out.add(queue.remove());
        }

        return out;
    }

    public int size(){
        return queue.size();
    }

    public int getCapacity() {
        return capacity;
    }
}
```

Tworzymy interfejs IMethodRequest, używany do przekazania kontekstu wywołania metody z Proxy do Schedulera uruchomionego w osobnym wątku.

```
public interface IMethodRequest {
    void call();
    boolean guard();
}
```

Implemenetujemy utworzony interfejs. Klasa AddRequest reprezentuje żądanie dodania listy liczb całkowitych do Servant. Do sprawdzania, czy operacja może być wykonana, służy metoda guard().

```
public class AddRequest implements IMethodRequest {
    private final Servant servant;
    private final List<Integer> object;
    private final Future future;
    private final long requestingThreadID;

    public AddRequest(Future future, Servant servant,
List<Integer> object) {
        this.future = future;
        this.servant = servant;
        this.object = object;
        this.requestingThreadID =
Thread.currentThread().getId();
    }

    @Override
    public void call() {
        System.out.println(this);
        servant.put(object);
        future.set(List.of(object.size()));
    }

    @Override
    public boolean guard() {
        return servant.size() + object.size() <=
servant.getCapacity();
    }

    @Override
    public String toString() {
        return "[AddRequest] Thread " +
                requestingThreadID +
                " - " +
                object.size() +
                " items";
    }
}
```

Klasa GetRequest również implementuje interfejs IMethodRequest. Reprezentuje ona żądanie pobrania określonej liczby elementów z Servant. Podobnie jak poprzednio, metoda guard() sprawdza, czy operacja pobierania może być wykonana.

```
public class GetRequest implements IMethodRequest {
    private final Servant servant;
    private final int elements;
    private final Future future;
```

```
    private final long requestingThreadID;

    public GetRequest(Future future, Servant servant, int
elements) {
        this.servant = servant;
        this.future = future;
        this.elements = elements;
        this.requestingThreadID =
Thread.currentThread().getId();
    }

    @Override
    public void call() {
        System.out.println(this);
        future.set(servant.get(elements));
    }

    @Override
    public boolean guard() {
        return servant.size() >= elements;
    }

    @Override
    public String toString() {
        return "[GetRequest] Thread " +
                requestingThreadID +
                " - " +
                elements +
                " items";
    }
}
```

Klasa Scheduler reprezentuje „planistę zadań”, implementowanych przez interfejs IMethodRequest. Planista ten działa w osobnym wątku, ciągle pobierając i obsługując zadania z kolejki. Jeżeli warunek guard dla danego zadania jest spełniony, to jest ono natychmiastowo wykonane (call). W przeciwnym razie zadanie pozostaje w kolejce, czekając na spełnienie warunku guard przed ponowną próbą jego wykonania.

```
public class Scheduler extends Thread{
    private final Queue<IMethodRequest> activationQueue = new
ConcurrentLinkedQueue<>();

    public void insert(IMethodRequest methodRequest){
        activationQueue.add(methodRequest);
    }

    @Override
    public void run(){
```

```
        while(true){
            var item = activationQueue.poll();
            if (item != null) {
                if (item.guard()) {
                    item.call();
                }else{
                    activationQueue.add(item);
                }
            }
        }
    }
}
```

Klasa Future dostarcza mechanizm do oczekiwania na zakończenie operacji i uzyskania wyniku. Wątki mogą używać metody await() do blokowania się, a następnie obudzić się, gdy operacja zostanie zakończona (za pomocą metody complete()).

```
public class Future {
    private List<Integer> object;
    private boolean completed = false;


    public void set(List<Integer> object) {
        this.object = object;
        complete();
    }

    public synchronized void complete(){
        completed = true;
        notify();
    }

    public boolean isCompleted(){
        return completed;
    }

    public synchronized void await(){
        while(!isCompleted()){
            try {
                wait();
            } catch (InterruptedException ignored) {
            }
        }
    }
}
```

Klasa Proxy służy jako warstwa pośrednicząca między klientem a obiektem Servant, umożliwiając asynchroniczne wykonywanie operacji na danych przy użyciu Schedulera. Operacje dodawania i pobierania są reprezentowane przez odpowiednie obiekty (AddRequest i GetRequest), które są przekazywane do Schedulera wraz z obiektami Future umożliwiającymi asynchroniczne oczekiwanie na zakończenie operacji i uzyskanie wyniku.

```
public class Proxy {
    private final Servant servant;
    private final Scheduler scheduler = new Scheduler();

    public Proxy(int bufferSize) {
        this.servant = new Servant(bufferSize);

        scheduler.setDaemon(true);
        scheduler.start();
    }

    public Future put(List<Integer> object) {
        var future = new Future();
        scheduler.insert(new AddRequest(future, servant,
object));
        return future;
    }

    public Future get(int numberOfElements) {
        var future = new Future();
        scheduler.insert(new GetRequest(future, servant,
numberOfElements));
        return future;
    }
}
```

Klasa Producer reprezentuje producenta, który korzysta z obiektu Proxy do asynchronicznego dodawania danych do obiektu Servant. Każdy producent wykonuje operacje dodawania w osobnym wątku, a po zakończeniu tych operacji czeka na zakończenie zadania, używając obiektów Future.

```
import java.util.LinkedList;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;

public class Producer extends Thread {
    private final Proxy proxy;
    private final int totalNumberToPut;
    private ThreadLocalRandom random;

    public Producer(Proxy proxy, int totalNumberToGet) {
        this.proxy = proxy;
```

```
        this.totalNumberToPut = totalNumberToGet;
    }

    public void _sleep(int milis){
        try{
            sleep(milis);
        }catch (InterruptedException ignored){
        }
    }

    public List<Integer> generateData(int elems){
        List<Integer> out = new LinkedList<>();
        while(elems > 0){
            out.add(random.nextInt(100));
            elems--;
        }
        return out;
    }

    @Override
    public void run(){
        random = ThreadLocalRandom.current();
        long id = Thread.currentThread().getId();

        List<Future> futureList = new LinkedList<>();
        int elementsLeft = totalNumberToPut;
        while(elementsLeft > 0) {
            int toPut = Math.min(elementsLeft,
random.nextInt(5) + 1);
            elementsLeft -= toPut;

            var future = proxy.put(generateData(toPut));
            futureList.add(future);

            _sleep(200);
        }

        System.out.println("[Producer] " + id + " finished
requesting!");
        for(var f : futureList){
            f.await();
        }
        System.out.println("[Producer] " + id + " finished
job!");
    }
}
```

Klasa Consumer reprezentuje konsumenta, który korzysta z obiektu Proxy do asynchronicznego pobierania danych od obiektu Servant. Każdy konsument wykonuje operacje pobierania w osobnym wątku, a po zakończeniu tych operacji czeka na zakończenie zadania, używając obiektów Future.

```
public class Consumer extends Thread {
    private final Proxy proxy;
    private final int totalNumberToGet;

    public Consumer(Proxy proxy, int totalNumberToGet) {
        this.proxy = proxy;
        this.totalNumberToGet = totalNumberToGet;
    }

    public void _sleep(int milis){
        try{
            sleep(milis);
        }catch (InterruptedException ignored){
        }
    }

    @Override
    public void run(){
        long id = Thread.currentThread().getId();

        List<Future> futureList = new LinkedList<>();
        int elementsLeft = totalNumberToGet;
        var random = ThreadLocalRandom.current();
        while(elementsLeft > 0) {
            int toGet = Math.min(elementsLeft,
random.nextInt(5) + 1);
            elementsLeft -= toGet;

            var future = proxy.get(toGet);
            futureList.add(future);

            _sleep(100);
        }

        System.out.println("[Consumer] " + id + " finished
requesting!");
        for(var f : futureList){
            f.await();
        }
        System.out.println("[Consumer] " + id + " finished
job!");
    }
}
```

Klasa Main inicjalizuje i uruchamia przykładowy scenariusz z użyciem wielu wątków producentów i konsumentów, którzy korzystają z obiektu Proxy do interakcji z obiektem Servant. W naszym przypadku liczba wątków producenta i konsumenta wynosi po 8, pojemność buffera wynosi 16, a każdy obiekt musi przetworzyć 32 elementy.

```
public class Main {
    private static final int BUFFER_CAPACITY = 16;
    private static final int CONSUMER_THREADS =8;
    private static final int PRODUCER_THREADS = 8;
    private static final int ELEMENTS_PER_THREAD = 32;


    public static void main(String[] args) {
        var threadList = new LinkedList<Thread>();
        var bufferProxy = new Proxy(BUFFER_CAPACITY);

        IntStream.range(0, CONSUMER_THREADS).forEach(i -> {
            threadList.add(new Consumer(bufferProxy,
ELEMENTS_PER_THREAD));
        });

        IntStream.range(0, PRODUCER_THREADS).forEach(i -> {
            threadList.add(new Producer(bufferProxy,
ELEMENTS_PER_THREAD));
        });

        threadList.forEach(Thread::start);

        threadList.forEach(t -> {
            try {
                t.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
    }
}
```

3. Wyniki

W wyniku uruchomienia programu otrzymujemy:

[AddRequest] Thread 33 - 4 items

[AddRequest] Thread 35 - 3 items

[GetRequest] Thread 26 - 3 items

[AddRequest] Thread 38 - 5 items

[GetRequest] Thread 24 - 5 items

[GetRequest] Thread 28 - 4 items

[AddRequest] Thread 37 - 1 items

[AddRequest] Thread 34 - 1 items

[AddRequest] Thread 31 - 2 items

[AddRequest] Thread 32 - 2 items

[AddRequest] Thread 36 - 2 items

[GetRequest] Thread 27 - 5 items

[GetRequest] Thread 23 - 1 items

[GetRequest] Thread 23 - 2 items

[AddRequest] Thread 38 - 5 items

[AddRequest] Thread 33 - 2 items

[GetRequest] Thread 29 - 5 items

[GetRequest] Thread 26 - 2 items

[AddRequest] Thread 32 - 1 items

[AddRequest] Thread 31 - 4 items

[GetRequest] Thread 29 - 3 items

[AddRequest] Thread 37 - 5 items

[AddRequest] Thread 36 - 1 items

[AddRequest] Thread 35 - 5 items

[GetRequest] Thread 24 - 3 items

[GetRequest] Thread 26 - 4 items

[GetRequest] Thread 25 - 4 items

[GetRequest] Thread 28 - 1 items

[AddRequest] Thread 34 - 4 items

[GetRequest] Thread 30 - 2 items

[GetRequest] Thread 30 - 3 items

[AddRequest] Thread 32 - 3 items

[GetRequest] Thread 30 - 2 items

[AddRequest] Thread 33 - 4 items

[AddRequest] Thread 35 - 3 items

[AddRequest] Thread 37 - 4 items

[GetRequest] Thread 29 - 5 items

[GetRequest] Thread 27 - 1 items

[GetRequest] Thread 26 - 4 items

[GetRequest] Thread 25 - 1 items

[AddRequest] Thread 36 - 5 items

[AddRequest] Thread 38 - 3 items

[AddRequest] Thread 34 - 5 items

[AddRequest] Thread 31 - 1 items

[GetRequest] Thread 23 - 4 items

[GetRequest] Thread 27 - 4 items

[GetRequest] Thread 27 - 4 items

[GetRequest] Thread 29 - 3 items

[AddRequest] Thread 36 - 1 items

[AddRequest] Thread 37 - 1 items

[GetRequest] Thread 27 - 2 items

[AddRequest] Thread 33 - 5 items

[AddRequest] Thread 35 - 4 items

[AddRequest] Thread 38 - 2 items

[AddRequest] Thread 32 - 4 items

[GetRequest] Thread 29 - 4 items

[GetRequest] Thread 26 - 5 items

[GetRequest] Thread 28 - 2 items

[GetRequest] Thread 27 - 3 items

[GetRequest] Thread 24 - 1 items

[AddRequest] Thread 31 - 3 items

[AddRequest] Thread 34 - 4 items

[GetRequest] Thread 28 - 2 items

[GetRequest] Thread 28 - 4 items

[GetRequest] Thread 23 - 1 items

[AddRequest] Thread 36 - 3 items

[AddRequest] Thread 38 - 5 items

[AddRequest] Thread 32 - 5 items

[AddRequest] Thread 31 - 1 items

[GetRequest] Thread 25 - 5 items

[AddRequest] Thread 37 - 5 items

[GetRequest] Thread 28 - 1 items

[GetRequest] Thread 29 - 5 items

[GetRequest] Thread 24 - 3 items

[GetRequest] Thread 24 - 3 items

[GetRequest] Thread 23 - 2 items

[AddRequest] Thread 35 - 5 items

[AddRequest] Thread 34 - 2 items

[AddRequest] Thread 33 - 3 items

[GetRequest] Thread 25 - 4 items

[GetRequest] Thread 23 - 5 items

[GetRequest] Thread 29 - 1 items

[Consumer] 25 finished requesting!

[AddRequest] Thread 33 - 1 items

[AddRequest] Thread 34 - 3 items

[GetRequest] Thread 23 - 3 items

[AddRequest] Thread 38 - 1 items

[GetRequest] Thread 23 - 1 items

[GetRequest] Thread 29 - 1 items

[AddRequest] Thread 36 - 1 items

[AddRequest] Thread 37 - 5 items

[AddRequest] Thread 35 - 4 items

[AddRequest] Thread 32 - 3 items

[GetRequest] Thread 28 - 4 items

[GetRequest] Thread 25 - 5 items

[GetRequest] Thread 29 - 3 items

[GetRequest] Thread 28 - 1 items

[AddRequest] Thread 31 - 5 items

[GetRequest] Thread 26 - 4 items

[GetRequest] Thread 28 - 1 items

[Consumer] 26 finished requesting!

[Consumer] 30 finished requesting!

[Consumer] 29 finished requesting!

[Consumer] 24 finished requesting!

[AddRequest] Thread 33 - 2 items

[GetRequest] Thread 27 - 1 items

[GetRequest] Thread 27 - 1 items

[AddRequest] Thread 34 - 3 items

[AddRequest] Thread 35 - 3 items

[AddRequest] Thread 31 - 3 items

[AddRequest] Thread 37 - 2 items

[AddRequest] Thread 36 - 2 items

[GetRequest] Thread 27 - 3 items

[GetRequest] Thread 26 - 3 items

[GetRequest] Thread 30 - 5 items

[GetRequest] Thread 29 - 2 items

[AddRequest] Thread 38 - 4 items

[AddRequest] Thread 32 - 4 items

[GetRequest] Thread 24 - 3 items

[GetRequest] Thread 30 - 3 items

[GetRequest] Thread 28 - 1 items

[GetRequest] Thread 27 - 1 items

[Consumer] 29 finished job!

[Consumer] 23 finished requesting!

[Consumer] 27 finished requesting!

[Consumer] 28 finished requesting!

[AddRequest] Thread 38 - 2 items

[AddRequest] Thread 31 - 3 items

[AddRequest] Thread 32 - 1 items

[AddRequest] Thread 36 - 3 items

[AddRequest] Thread 33 - 2 items

[AddRequest] Thread 34 - 5 items

[GetRequest] Thread 28 - 3 items

[GetRequest] Thread 26 - 2 items

[AddRequest] Thread 35 - 4 items

[GetRequest] Thread 26 - 2 items

[GetRequest] Thread 23 - 2 items

[GetRequest] Thread 24 - 5 items

[GetRequest] Thread 28 - 5 items

[GetRequest] Thread 24 - 1 items

[AddRequest] Thread 37 - 4 items

[GetRequest] Thread 28 - 3 items

[Consumer] 28 finished job!

[AddRequest] Thread 34 - 3 items

[AddRequest] Thread 36 - 4 items

[AddRequest] Thread 37 - 1 items

[AddRequest] Thread 35 - 1 items

[GetRequest] Thread 25 - 3 items

[AddRequest] Thread 31 - 5 items

[GetRequest] Thread 27 - 4 items

[GetRequest] Thread 30 - 2 items

[GetRequest] Thread 24 - 3 items

[GetRequest] Thread 30 - 3 items

[AddRequest] Thread 32 - 1 items

[AddRequest] Thread 33 - 4 items

[AddRequest] Thread 38 - 5 items

[GetRequest] Thread 25 - 2 items

[GetRequest] Thread 25 - 4 items

[GetRequest] Thread 30 - 4 items

[AddRequest] Thread 33 - 1 items

[AddRequest] Thread 32 - 3 items

[AddRequest] Thread 31 - 2 items

[AddRequest] Thread 36 - 1 items

[AddRequest] Thread 37 - 1 items

[GetRequest] Thread 26 - 3 items

[GetRequest] Thread 23 - 3 items

[AddRequest] Thread 34 - 2 items

[Consumer] 26 finished job!

[GetRequest] Thread 27 - 3 items

[Consumer] 27 finished job!

[Producer] 35 finished requesting!

[Producer] 38 finished requesting!

[Producer] 35 finished job!

[Producer] 38 finished job!

[Producer] 34 finished requesting!

[AddRequest] Thread 33 - 4 items

[Producer] 34 finished job!

[GetRequest] Thread 24 - 5 items

[AddRequest] Thread 31 - 1 items

[Consumer] 24 finished job!

[AddRequest] Thread 36 - 5 items

[GetRequest] Thread 25 - 4 items

[AddRequest] Thread 37 - 2 items

[GetRequest] Thread 30 - 3 items

[AddRequest] Thread 32 - 3 items

[Consumer] 25 finished job!

[GetRequest] Thread 23 - 3 items

[Producer] 33 finished requesting!

[AddRequest] Thread 36 - 1 items

[Producer] 33 finished job!

[AddRequest] Thread 37 - 1 items

[AddRequest] Thread 31 - 2 items

[AddRequest] Thread 32 - 2 items

[GetRequest] Thread 30 - 5 items

[Consumer] 30 finished job!

[Producer] 31 finished requesting!

[Producer] 32 finished requesting!

[Producer] 37 finished requesting!

[AddRequest] Thread 36 - 3 items

[Producer] 31 finished job!

[Producer] 32 finished job!

[Producer] 37 finished job!

[GetRequest] Thread 23 - 5 items

[Consumer] 23 finished job!

[Producer] 36 finished requesting!

[Producer] 36 finished job!

Analizując te wyniki, można zauważyć, że operacje dodawania i pobierania zachodzą współbieżnie, a wątki konsumentów i producentów wykonują operacje równocześnie. Komunikaty finished requesting! i finished job! wskazują na zakończenie operacji przez poszczególne wątki.

Sam program działa prawidłowo. Wątki dodają polecenia do Schedulera, a ten wykonuje zadania. Operacje, które nie mogą zostać wykonane, zostają dodane na koniec kolejki.

Warto zauważyć, że wyniki mogą się różnić przy różnych uruchomieniach programu, ponieważ są one zależne od harmonogramu planisty wątków oraz równoległego wykonywania operacji.

4. Wnioski

- Wzorzec Active Object pozwala na wyodrębnienie operacji asynchronicznych, umożliwiając wykonywanie ich w tle, co może być korzystne w przypadku operacji, które wymagają czasochłonnego przetwarzania.
- Active Object umożliwia asynchroniczne wywołanie operacji bez blokady głównego wątku, co pozwala na efektywne wykorzystanie zasobów i zwiększenie responsywności systemu. Wykorzystywane jest to w systemie Android, gdzie uaktualnienie widoków aplikacji następuje po zakończeniu obliczeń w Schedulerze.
- Wzorzec ten pomaga w uniezależnieniu producentów i konsumentów od siebie, co może być szczególnie użyteczne w systemach, gdzie niezbędne jest zachowanie elastyczności i równowagi pomiędzy różnymi rodzajami operacji.
- Wzorzec ten ułatwia rozszerzalność systemu, ponieważ nowe operacje asynchroniczne mogą być dodawane bez konieczności ingerencji w kod istniejący.
- Wadą wzorca jest trudność w debugowaniu programu, gdyż jest on wielowątkowy, a sam Scheduler może dodatkowo szeregować zadania w różnej kolejności.

5. Bibliografia

(Każdy podpunkt jest hiperłączem)

- Materiały do laboratorium
- Quantum Leaps – Active Object
- Android and Java Concurrency: The Active Object Pattern
- Dokumentacja Javy