TP 4 - Producteur / consommateur

Exercice 1 - Exercice 1 - Producer / Consumer

1. Le design pattern **Producer/Consumer** permet de résoudre le problème lorsqu'il y a plus de **clients** que de **threads** ou moins de **clients** que de **threads**

2. Pour mettre une valeur dans le buffer c'est **put()** et retirer **take()***

3.
```java
private static Runnable producer(int id, int timestamp) {
    return () -> {
        for (;;) {
            try {
                System.out.println("hello " + id);
                Thread.sleep(timestamp);
            } catch (InterruptedException e) {
                return;
            }
        }
    };
}

public static void main(String[] args) {
    var threads = new ArrayList<Thread>();
    var timestamps = List.of(1, 4);
    var nbThread = 2;

    IntStream.range(0, nbThread).forEach(i -> {
        threads.add(new Thread(producer(i, timestamps.get(i))));
    });
    for (var i = 0; i < nbThread; i++) {
        threads.get(i).start();
    }
}
```

4. Lorsque l'on utilise une LinkedBlockingQueue et que l'on ne lui passe pas de taille en paramètre, celui-ci plante lorsqu'il n'y a plus assez de mémoire.

```java
private static Runnable producer(BlockingQueue<String> bQueue, int id, int
timestamp) {
    return () -> {
        for (;;) {
            try {
                bQueue.put("hello " + id);
                Thread.sleep(timestamp);
            } catch (InterruptedException e) {
                return;
            }
        }
    };
}
```

```
public static void main(String[] args) {
    var threads = new ArrayList<Thread>();
    var timestamps = List.of(1, 4);
    var nbThread = 2;

    var arrayBQueue = new ArrayBlockingQueue<String>(nbThread);

    IntStream.range(0, nbThread).forEach(i -> {
        threads.add(new Thread(producer(arrayBQueue, i, timestamps.get(i))));
    });
    for (var i = 0; i < nbThread; i++) {
        threads.get(i).start();
    }
    for (var i = 0; i < nbThread; i++) {
        threads.get(i).start();
    }
}

public static void main(String[] args) {
    var threads = new ArrayList<Thread>();
    var nbThread = 2;

    var linkedBQueue = new LinkedBlockingQueue<String>();

    IntStream.range(0, nbThread).forEach(i -> {
        threads.add(new Thread(producer(linkedBQueue, i, 0)));
    });
    for (var i = 0; i < nbThread; i++) {
        threads.get(i).start();
    }
}
```

5.

```
 private static Runnable producer(BlockingQueue<String> bQueue, int id, int
timestamp) {
     return () -> {
         for (;;) {
             try {
                 bQueue.put("hello " + id);
                 Thread.sleep(timestamp);
             } catch (InterruptedException e) {
                 return;
             }
         }
     };
 }

 private static Runnable consumer(BlockingQueue<String> bQueue) {
     return () -> {
         for (;;) {
             try {
                 System.out.println(bQueue.take());
             } catch (InterruptedException e) {
                return;
            }
        }
    };
}

public static void main(String[] args) {
    var threads = new ArrayList<Thread>();
    var timestamps = List.of(2, 3, 5, 12, 19);
    var nbThread = 5;

    var arrayBQueue = new ArrayBlockingQueue<String>(nbThread);

    // producer
    IntStream.range(0, nbThread).forEach(i -> {
        threads.add(new Thread(producer(arrayBQueue, i, timestamps.get(i))));
    });

    for (var i = 0; i < nbThread; i++) {
        threads.get(i).start();
    }

    // consumer
    IntStream.range(0, nbThread).forEach(i -> {
        new Thread(consumer(arrayBQueue)).start();
    });
}
```

Exercice 2 - Exercice 2 - Queue bloquante

```
public class SynchronizedBlockingBuffer {
    private final ArrayDeque<String> buffer;
    private final int capacity;

    public SynchronizedBlockingBuffer(int capacity) {
        if (capacity < 1) {
            throw new IllegalArgumentException("capacity < 1");
        }
        this.capacity = capacity;
        this.buffer = new ArrayDeque<>(capacity);
    }

    public void put(String message) throws InterruptedException {
        synchronized (buffer) {
            while (buffer.size() == capacity) {
                buffer.wait();
            }
            buffer.addLast(message);
            buffer.notifyAll();
        }
    }
```

```java
    public String take() throws InterruptedException {
        synchronized (buffer) {
            while (buffer.size() == 0) {
                buffer.wait();
            }
            buffer.notifyAll();
            return buffer.removeFirst();
        }
    }
}
```

```java
public class LockedBlockingBuffer {
    private final ArrayDeque<String> buffer;
    private final ReentrantLock lock = new ReentrantLock();
    private final Condition isEmpty = lock.newCondition();
    private final Condition isFull = lock.newCondition();
    private final int capacity;

    public LockedBlockingBuffer(int capacity) {
        if (capacity < 1) {
            throw new IllegalArgumentException("capacity < 1");
        }
        this.capacity = capacity;
        this.buffer = new ArrayDeque<>(capacity);
    }

    public void put(String message) throws InterruptedException {
        lock.lock();
        try {
            while (buffer.size() == capacity) {
                isFull.await();
            }
            buffer.addLast(message);
            isEmpty.signalAll();
        } finally {
            lock.unlock();
        }
    }

    public String take() throws InterruptedException {
        lock.lock();
        try {
            while (buffer.size() == 0) {
                isEmpty.await();
            }
            isFull.signalAll();
            return buffer.removeFirst();
        } finally {
            lock.unlock();
        }
    }
}
```

Steve Chen 29/10/2021