Chương 1: Tổng quan

* 1. Khái niệm chung về ConcurrentLinkedQueue
     1. Lớp ConcurrentLinkedQueue trong Java là một phần của Java Collection Framework. Nó thuộc về gói java.util.concurrent. Nó đã được giới thiệu trong JDK 1.5. Nó được sử dụng để implement Queue với sự trợ giúp của LinkedList. Đây là một triển khai hàng đợi an toàn theo luồng không bị ràng buộc, nó sẽ chèn các phần tử vào cuối “ hàng đợi ” theo kiểu FIFO (nhập trước xuất trước). Nó có thể được sử dụng khi một “ hàng đợi ” không bị ràng buộc được chia sẻ giữa nhiều luồng. Lớp này không cho phép các phần tử rỗng. Các trình lặp có tính nhất quán yếu. Lớp này và iterator của nó thực thi tất cả các phương thức tùy chọn của giao diện Queue và Iterator.
  2. ConcurrentLinkedQueue trong java
     1. Cách khai báo trong ConcurrentLinkedQueue

**public** **class** ConcurrentLinkedQueue<E> **extends** AbstractQueue<E>

**implements** Queue<E>, java.io.Serializable {

// methods ...

}

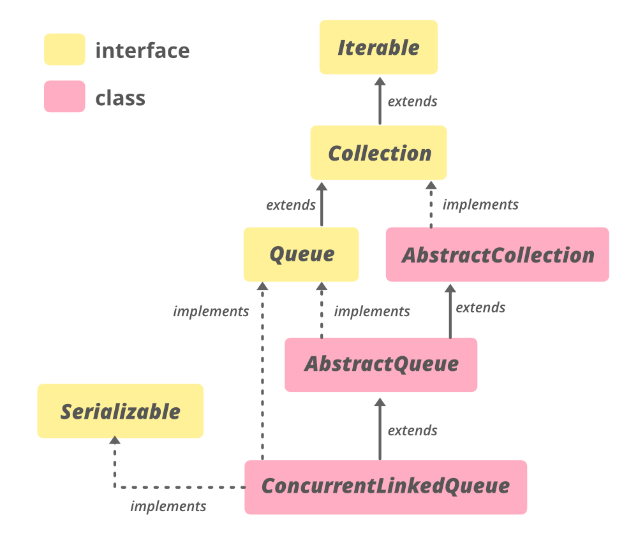
* + 1. Hệ thống phân cấp lớp

java.lang.Object

↳ java.util.AbstractCollection<E>

↳ java.util.AbstractQueue<E>

↳ Class ConcurrentLinkedQueue<E>



* + 1. Cách tạo một ConcurrentLinkedQueue
* Constructor dùng để khởi tạo một hàng đợi trống:

ConcurrentLinkedQueue<E> clq = **new** ConcurrentLinkedQueue<E>();

* Constructor dùng để khởi tạo một hàng đợi với những phần tử được truyền vào dưới dạng các tham số:

ConcurrentLinkedQueue<E> clq = **new** ConcurrentLinkedQueue<E>(Collection<E> c);

* + 1. Thêm phần tử
* Thêm một phần tử: phương thức add () hoặc offer() nó chèn phần tử, được truyền dưới dạng tham số ở đuôi ConcurrentLinkedQueue này. Phương thức này trả về True nếu chèn thành công. ConcurrentLinkedQueue là không bị ràng buộc, vì vậy phương thức này sẽ không bao giờ ném IllegalStateException hoặc trả về false.

ConcurrentLinkedQueue<E> concurrentLinkedQueue =

**new** ConcurrentLinkedQueue<E>();

concurrentLinkedQueue.add(e);

// or

concurrentLinkedQueue.offer(e);

* Thêm nhiều phần tử: addAll () Nó chèn tất cả các phần tử của Collection, được truyền dưới dạng tham số ở cuối ConcurrentLinkedQueue. Việc chèn phần tử theo thứ tự như được trả về bởi trình vòng lặp của Collection .

ConcurrentLinkedQueue<E> concurrentLinkedQueue = **new**

ConcurrentLinkedQueue<E>();

ArrayList<E> list = **new** ArrayList<>();

concurrentLinkedQueue.addAll(list);

* + 1. Lấy giá trị
* Hiển thị phần tử đầu tiên sử dụng phương thức peek(). Phương thức này lấy phần tử đầu tiên và không xóa phần tử đó ra khỏi hàng đợi.

ConcurrentLinkedQueue<E> concurrentLinkedQueue = **new**

ConcurrentLinkedQueue<E>();

concurrentLinkedQueue.peek();

* Phương thức element() hiển thị phần tử đầu tiên trong hàng đợi nhưng không xóa .

ConcurrentLinkedQueue<E> concurrentLinkedQueue = **new**

ConcurrentLinkedQueue<E>();

concurrentLinkedQueue.element();

* Hiển thị phần tử đầu tiên sử dụng phương thức poll(). Phương thức này xóa và hiển thị phần tử đầu tiên.

ConcurrentLinkedQueue<E> concurrentLinkedQueue = **new**

ConcurrentLinkedQueue<E>();

concurrentLinkedQueue.poll();

* + 1. Xóa giá trị
* Phương thức remove (Object o) của ConcurrentLinkedQueue được sử dụng để loại bỏ một trường hợp duy nhất của phần tử được chỉ định nếu nó có mặt. Nó loại bỏ một phần tử e sao cho o.equals (e). Nó trả về true nếu ConcurrentLinkedQueue này chứa phần tử được chỉ định khác, nó sẽ trả về false.

ConcurrentLinkedQueue<E> concurrentLinkedQueue = **new**

ConcurrentLinkedQueue<E>();

Object object = **new** Object();

concurrentLinkedQueue.remove(object);

* + 1. Các phần tử lặp
* Phương thức iterator () của ConcurrentLinkedQueue được sử dụng để trả về một trình lặp của các phần tử giống như ConcurrentLinkedQueue này theo một trình tự thích hợp. Các phần tử được trả về từ phương thức này chứa các phần tử theo thứ tự từ đầu tiên (đầu) đến cuối cùng (đuôi). Trình lặp được trả về là nhất quán yếu.

ConcurrentLinkedQueue<E> concurrentLinkedQueue = **new**

ConcurrentLinkedQueue<E>();

// Call iterator() method

Iterator<E> iterator = concurrentLinkedQueue.iterator();

// Print elements of iterator

**while** (iterator.hasNext()) {

// value = iterator.next();

}

Chương 2 Các phương thức của ConcurrentLinkedQueue và cách sử dụng

2.1 Phương thước add​(E e) .

- Phương thức add () của ConcurrentLinkedQueue được sử dụng để chèn phần tử, được truyền dưới dạng tham số để add () của ConcurrentLinkedQueue, ở phần đuôi của ConcurrentLinkedQueue này. Phương thức này trả về True nếu chèn thành công. ConcurrentLinkedQueue là không bị ràng buộc, vì vậy phương thức này sẽ không bao giờ ném IllegalStateException hoặc trả về false.

- Code

/\*\*

\* - Phương thức <b>thêm</b> một chuỗi str vào hàng đợi queue.</br>

\* - Phương thức này sẽ chèn một chuỗi str vào cuối của hàng đợi.</br>

\* - Phương thức này sẽ trả về true nếu thêm thành công và nó sẽ trả về false

\* nếu thêm thất bại.

\*

\* **@param** queue

\* **@param** str

\* **@return**

\*/

**public** **static** **boolean** add(ConcurrentLinkedQueue<String> queue, String str) {

// Phương thức add() sẽ chèn phần tử được chỉ định vào cuối của hàng

// đợi

**return** queue.add(str);

}

2.2 Phương thức addAll​(Collection<? extends E> c) .

Phương thức addAll () của ConcurrentLinkedQueue được sử dụng để chèn tất cả các phần tử của Collection, được truyền dưới dạng tham số cho phương thức này, vào cuối ConcurrentLinkedQueue. Việc chèn phần tử theo thứ tự như được trả về bởi collections iterator.

2.2.1 Phương thức thêm từ danh sách ArrayList.

/\*\*

\* - Phương thức <b>thêm</b> một danh sách chuỗi vào hàng đợi.</br>

\* - Phương thức này sẽ chèn một danh sách String vào cuối của hàng đợi.</br>

\* - Phương thức này sẽ trả về true nếu thêm thành công và nó sẽ trả về false

\* nếu thêm thất bại..

\*

\* **@param** queue

\* **@param** listString

\* **@return**

\*/

**public** **static** **boolean** addAll(ConcurrentLinkedQueue<String> queue,

ArrayList<String> listString) {

// Phương thức addAll() sẽ chèn tất cả các phần tử của danh sách

// vào cuối của hàng đợi.

**return** queue.addAll(listString);

}

2.2.2 Phương thức thêm từ ConcurrentLinkedQueue.

/\*\*

\* - Phương thức <b>thêm</b> một hàng đợi newQueue vào một hàng đợi khác

\* queue.</br>

\* - Phương thức này sẽ chèn một hàng đợi newQueue vào cuối của hàng đợi

\* queue.</br>

\* - Phương thức này sẽ trả về true nếu thêm thành công và nó sẽ trả về false

\* nếu thêm thất bại.

\*

\* **@param** queue

\* **@param** newQueue

\* **@return**

\*/

**public** **static** **boolean** addAll(ConcurrentLinkedQueue<String> queue,

ConcurrentLinkedQueue<String> newQueue) {

// Phương thức addAll() sẽ chèn tất cả các phần tử của newQueue vào

// cuối của hàng đợi queue.

**return** queue.addAll(newQueue);

}

2.3 Phương thức contains​(Object o) .

- Phương thức contains() của ConcurrentLinkedQueue trả về true nếu ConcurrentLinkedQueue chứa đối tượng o, được truyền dưới dạng tham số. Phương thức này trả về true nếu và chỉ khi ConcurrentLinkedQueue này chứa ít nhất một phần tử e bằng đối tượng o được truyền dưới dạng tham số, tức là o.equals (e).

- Code :

2.4 Phương thức forEach​(Consumer<? super E> action).

- Đây là phương thức dùng để đọc thông tin của từng phần tử trong hàng đợi.

- Code :

/\*\*

\* Phương thức in hàng đợi ra màn hình sử dụng forEach().

\* **@param** queue

\*/

**public** **static** **void** printQueueByForEach(ConcurrentLinkedQueue<String> queue) {

queue.forEach(element->{

// In từng phần tử trong hàng đợi ra màn hình

System.***out***.println(element);

});

}

2.5 Phương thức isEmpty() .

- Phương thức isEmpty () của ConcurrentLinkedQueue được sử dụng để kiểm tra xem hàng đợi này có trống hay không. Nó trả về true nếu ConcurrentLinkedQueue chứa số phần tử bằng không có nghĩa là nếu ConcurrentLinkedQueue trống.

- Code:

/\*\*

\* Phương thức kiểm tra xem hàng đợi có rỗng không</br>

\* -> Trả về true nếu hàng đợi không chứa phần tử.

\* **@param** queue

\* **@return**

\*/

**public** **static** **boolean** isEmpty(ConcurrentLinkedQueue<String> queue) {

**return** queue.isEmpty();

}

2.6 Phương thức iterator().

- Phương thức iterator () của ConcurrentLinkedQueue được sử dụng để trả về một trình lặp của các phần tử giống như ConcurrentLinkedQueue này theo một trình tự thích hợp. Các phần tử được trả về từ phương thức này chứa các phần tử theo thứ tự từ đầu tiên (đầu) đến cuối cùng (đuôi). Trình lặp được trả về là nhất quán yếu.

- Code :

/\*\*

\* Phương thức in từng phần tử trong queue ra màn hình sử dụng iterator().

\* **@param** queue

\*/

**public** **static** **void** printQueueByIterator(ConcurrentLinkedQueue<String> queue) {

// Call iterator() method

Iterator<String> iterator = queue.iterator();

// Print elements of iterator

System.***out***.println("\nThe String Values of iterator are:");

**while** (iterator.hasNext()) {

System.***out***.println(iterator.next());

}

}

2.7 Phương thức offer​(E e) .

- Phương thức offer () của ConcurrentLinkedQueue được sử dụng để chèn phần tử, được truyền dưới dạng tham số, vào đuôi ConcurrentLinkedQueue này. Phương thức này trả về True nếu chèn thành công. ConcurrentLinkedQueue là không bị ràng buộc, vì vậy phương thức này offer () sẽ không bao giờ trả về false.

- Code:

/\*\*

\* - Phương thức <b>thêm</b> một chuỗi str vào hàng đợi queue.</br>

\* - Phương thức này sẽ chèn một chuỗi str vào cuối của hàng đợi.</br>

\* - Phương thức này sẽ trả về true nếu thêm thành công và nó sẽ trả về false

\* nếu thêm thất bại.

\* **@param** queue

\* **@param** str

\* **@return**

\*/

**public** **static** **boolean** addByOffer(ConcurrentLinkedQueue<String> queue, String

str) {

// Phương thức offer() sẽ chèn phần tử được chỉ định vào cuối của hàng

// đợi

**return** queue.offer(str);

}

2.8 Phương thức remove​(Object o) .

- Phương thức remove (Object o) của ConcurrentLinkedQueue được sử dụng để loại bỏ một trường hợp duy nhất của phần tử được chỉ định khỏi ConcurrentLinkedQueue này, nếu nó có mặt. Phương thức này loại bỏ một phần tử e sao cho o.equals (e) nếu ConcurrentLinkedQueue này chứa một hoặc nhiều phần tử như vậy. Phương thức Remove () trả về true nếu ConcurrentLinkedQueue này chứa phần tử được chỉ định nếu không nó sẽ trả về false.

- Code :

/\*\*

\* Phương thức xóa phần tử có giá trị value trong hàng đợi queue.

\* **@param** queue

\* **@param** value

\* **@return**

\*/

**public** **static** **boolean** removeElement(ConcurrentLinkedQueue<String> queue,

String value) {

**return** queue.remove(value);

}

2.9 Phương thức removeAll​(Collection<?> c) .

- Loại bỏ tất cả các phần tử của Collection này cũng có trong Collection được chỉ định (thao tác tùy chọn).

- Phương thức loại bỏ tất cá các phần tử trong queue theo ArrayList.

/\*\*

\* Phương thức xóa tất cả các phần tử của hàng đợi queue có trong danh sách

\* listRemove.

\* **@param** queue

\* **@param** listRemove

\* **@return**

\*/

**public** **static** **boolean** removeAllByList(ConcurrentLinkedQueue<String> queue,

ArrayList<String> listRemove) {

**return** queue.removeAll(listRemove);

}

- Phương thức loại bỏ tất các các phần tử trong queue theo một hàng đợi cho trước:

/\*\*

\* Phương thức xóa tất các các phần tử của ngăn xếp queue có trong ngăn xếp

\* queueRemove.

\* **@param** queue

\* **@param** queueRemove

\* **@return**

\*/

**public** **static** **boolean** removeAllByQueue(ConcurrentLinkedQueue<String> queue,

ConcurrentLinkedQueue<String> queueRemove) {

**return** queue.removeAll(queueRemove);

}

2.10 Phương thức removeIf​(Predicate<? super E> filter) .

- Loại bỏ tất cả các phần tử của tập hợp này thỏa mãn vị từ đã cho.

/\*\*

\* Phương thức xóa giá trị trong hàng đợi với điều kiện xác định.

\* **@param** queue

\* **@param** valueRemove

\* **@return**

\*/

**public** **static** **boolean** removeIf(ConcurrentLinkedQueue<String> queue,String

valueRemove) {

// Phương thức xóa các phần tử trong hàng đợi với điều kiện.

**return** queue.removeIf(value -> value.equals(valueRemove));

}

2.11 Phương thức spliterator() .

- Phương thức spliterator () của ConcurrentLinkedQueue được sử dụng để lấy một Spliterator có các phần tử giống như ConcurrentLinkedQueue. Đã tạo Spliterator nhất quán yếu. Nó có thể được sử dụng với Streams trong Java 8. Ngoài ra, nó cũng có thể duyệt các phần tử riêng lẻ và hàng loạt. Spliterator là cách tốt hơn để duyệt qua phần tử vì nó cung cấp nhiều quyền kiểm soát hơn trên các phần tử.

- Code :

/\*\*

\* Phương thức in từng phần tử trong queue ra màn hình sử dụng spliterator().

\*

\* **@param** queue

\*/

**public** **static** **void** printQueueBySpliterator(ConcurrentLinkedQueue<String>

queue) {

// create Spliterator of ConcurrentLinkedQueue

// using spliterator() method

Spliterator<String> spt = queue.spliterator();

// print result from Spliterator

System.***out***.println("List of ConcurrentLinkedQueue:");

// forEachRemaining method of Spliterator

spt.forEachRemaining((n) -> System.***out***.println(n + ", "));

}

2.11 Phương thức toArray() .

- Phương thức toArray () của ConcurrentLinkedQueue được sử dụng để trả về một mảng có các phần tử giống như của ConcurrentLinkedQueue theo trình tự thích hợp. Về cơ bản, nó sao chép tất cả phần tử từ ConcurrentLinkedQueue sang một mảng mới. Phương thức này hoạt động như một cầu nối giữa mảng và ConcurrentLinkedQueue.

- Code :

/\*\*

\* Phương thức chuyển hàng đợi thành mảng một chiều sử dụng phương thức

\* toArray();

\* **@param** queue

\* **@return**

\*/

**public** **static** Object [] toArray(ConcurrentLinkedQueue<String> queue){

// Phương thức toArray() dùng để chuyển một hàng đợi thành mảng đối

// tượng.

Object[] obj = queue.toArray();

**return** obj;

}

2.12 Phương thức hiển lấy phần tử trong ConcurrentLinkedQueue;

2.12.1 Phương thức peek()

- Phương thưc peek() dùng để lấy phần tử đầu tiên trong hàng đợi và không xóa phần tử đó ra khỏ hàng đợi.

/\*\*

\* Phương thức lấy phần tử đầu tiên trong hàng đợi sử dụng phương thức peek().

\* **@param** queue

\* **@return**

\*/

**public** **static** String getValueHeadByPeek(ConcurrentLinkedQueue<String> queue) {

// Phương thức peek() dùng để lấy phần tử đầu tiên trong hàng đợi.

// Không xóa phần tử đó ra khỏi hàng đợi.

**return** queue.peek();

}

2.12.2 Phương thức element().

- Phương thức element() dùng để lấy phần tử đầu tiên trong hàng đợi và không xóa phần tử đó ra khỏi hàng đợi,

/\*\*

\* Phương thức lấy phần tử đầu tiên trong hàng đợi sử dụng phương thức

\* element().

\* **@param** queue

\* **@return**

\*/

**public** **static** String getValueHeadByElement(ConcurrentLinkedQueue<String>

queue) {

// Phương thức element() dùng để lấy phần tử đầu tiên trong hàng đợi.

// Không xóa phần tử đó ra khỏi hàng đợi.

**return** queue.element();

}

2.12.3 Phương thức poll().

- Phương thức poll() dùng để lấy phần tử đầu tiên trong hàng đợi và xóa nó ra khỏi hàng đợi.

/\*\*

\* Phương thức lấy phần tử đầu tiên trong hàng đợi và xóa giá trị đó ra

\* khỏi hàng đợi sử dụng phương thức poll().

\* **@param** queue

\* **@return**

\*/

**public** **static** String getValueHeadByPoll(ConcurrentLinkedQueue<String>

queue) {

// Phương thức poll() dùng để lấy phần tử đầu tiên trong hàng đợi.

// Và xóa phần tử đó ra khỏi hàng đợi.

**return** queue.poll();

}

Chương 3 Kịch bản chạy và kết quả

3.1 Kịch bản chạy cho phương thức khởi tạo với constructor

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Create a ConcurrentLinkedQueue

// using ConcurrentLinkedQueue() contructor

ConcurrentLinkedQueue<String> clq = **new**

ConcurrentLinkedQueue<String>();

clq.add("A");

clq.add("B");

clq.add("C");

clq.add("D");

// Displaying the existing LinkedQueue

System.***out***.println("ConcurrentLinkedQueue: " + clq);

// Create a ConcurrentLinkedQueue

// using ConcurrentLinkedQueue(Collection c)

// constructor

ConcurrentLinkedQueue<String> clq1 = **new**

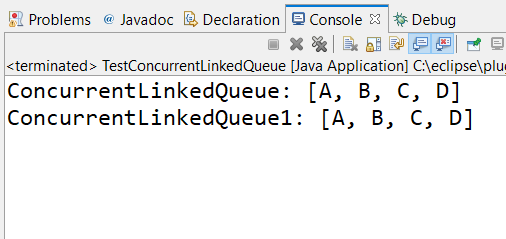
ConcurrentLinkedQueue<String>(clq);

// Displaying the existing LinkedQueue

System.***out***.println("ConcurrentLinkedQueue1: " + clq1);

}

- Kết quả chạy:



3.2 Kịch bản chạy cho phương thức add()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq = **new**

ConcurrentLinkedQueue<String>();

//Sử dụng phương thức add

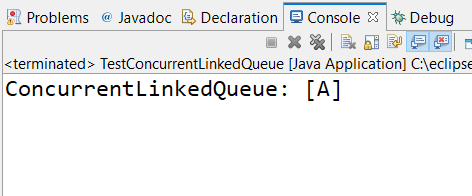
MyConcurrentLinkedQueue.*add*(clq, "A");

// Hiển thị ConcurrentLinkedQueue hiện có

System.***out***.println("ConcurrentLinkedQueue: " + clq);

}

- Kết quả chạy:



3.3 Kịch bản chạy cho phương thức addAll()

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq = **new**

ConcurrentLinkedQueue<String>();

// Sử dụng phương thức addAll() theo ArrayList();

// Tạo một ArrayList

ArrayList<String> temp = **new** ArrayList<String>(List.*of*("A", "B", "C"));

// Phương thức addAll từ một ArrayList

MyConcurrentLinkedQueue.*addAll*(clq, temp);

// Hiển thị ConcurrentLinkedQueue hiện có

System.***out***.println("ConcurrentLinkedQueue: " + clq);

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq2 = **new**

ConcurrentLinkedQueue<String>();

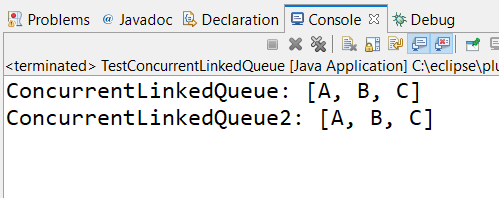
// Phương thức addAll từ một ConcurrentLinkedQueue

MyConcurrentLinkedQueue.*addAll*(clq2, clq);

System.***out***.println("ConcurrentLinkedQueue2: "+ clq2);

}

- Kết quả chạy:



3.4 Kịch bản chạy cho phương thức [contains​​](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/concurrentlinkedqueue-contains-method-in-java/&sa=U&ved=2ahUKEwjvltatsvLrAhUoxTgGHba8BRkQFjAGegQIAhAC&usg=AOvVaw2qDY3VWUvoPmZro4X5mznF)()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// Hiển thị ConcurrentLinkedQueue hiện có

System.***out***.println(clq);

// Dùng phương thức contains() để tìm kiếm phần tử trong hàng đợi

**if**(MyConcurrentLinkedQueue.*search*(clq, "One")) {

System.***out***.println(**true**);

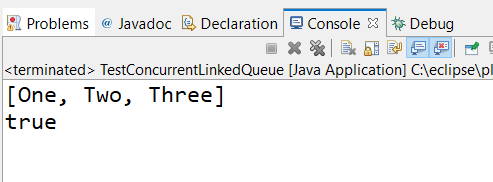
}**else** {

System.***out***.println(**false**);

}

}

- Kết quả chạy:



3.5 Kịch bản chạy cho phương thức forEach()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

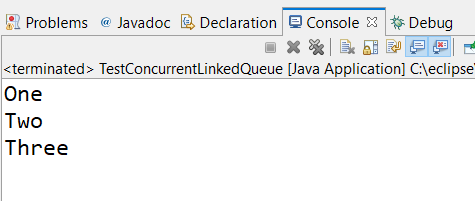
// Phương thức in thông tin của các phần tử trong clq ra màn hình

// sử dụng forEach

MyConcurrentLinkedQueue.*printQueueByForEach*(clq);

}

- Kết quả đạt được:



3.6 Kịch bản chạy cho phương thức isEmpty()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq = **new**

ConcurrentLinkedQueue<String>();

// Kiểm tra xem hàng đợi trên có rỗng hay không sử dụng phương thức

// isEmpty().

// Nếu rỗng thì trả về true và ngước lại.

**if**(MyConcurrentLinkedQueue.*isEmpty*(clq)) {

System.***out***.println(**true**);

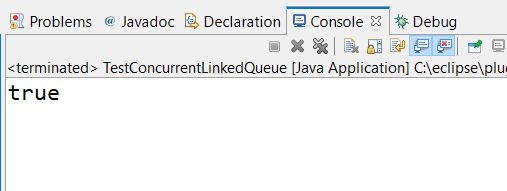
}**else** {

System.***out***.println(**false**);

}

}

- Kết quả chạy:



3.7 Kịch bản chạy cho phương thức interator()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

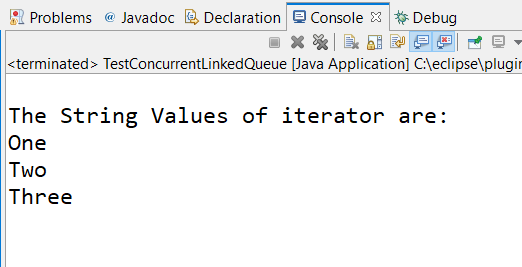
MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// In clq sử dụng interator()

MyConcurrentLinkedQueue.*printQueueByIterator*(clq);

}

- Kết quả chạy:



3.8 Kịch bản chạy cho phương thức offer()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq = **new**

ConcurrentLinkedQueue<String>();

// Sử dụng offer() để thêm phần tử vào clq

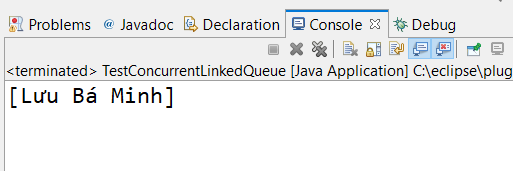
MyConcurrentLinkedQueue.*addByOffer*(clq, "Lưu Bá Minh");

// Hiển thị hàng đợi hiện có

System.***out***.println(clq);

}

- Kết quả chạy:



3.9 Kịch bản chạy cho phương thức remove()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// Hiển thị hàng đợi hiện có

System.***out***.println(clq);

// Xóa phần tử khỏi hàng đợi

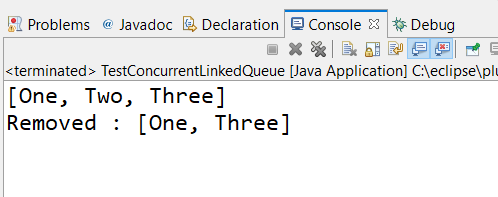
MyConcurrentLinkedQueue.*removeElement*(clq, "Two");

// Hiển thị hàng đợi sau khi đã xóa một phần tử

System.***out***.println("Removed : "+clq);

}

- Kết quả chạy



3.10 Kịch bản chạy cho phương thức removeAll()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// Hiển thị hàng đợi hiện có

System.***out***.println(clq);

// Tạo một danh sách cần xóa

ArrayList<String> listRemove = **new**

ArrayList<String>(List.*of*("One","Three"));

// Dùng removeAll() để xóa danh sách listRemove ra khỏi hàng đợi clq

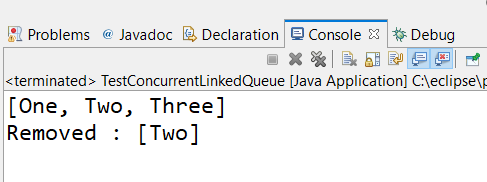
MyConcurrentLinkedQueue.*removeAllByList*(clq, listRemove);

// Hiển thị hàng đợi sau khi xóa

System.***out***.println("Removed : "+ clq);

}

- Kết quả chạy



3.11 Kịch bản chạy cho phương thức removeIf().

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// Hiển thị hàng đợi hiện có

System.***out***.println(clq);

// Tạo một danh sách cần xóa

// Dùng removeIf() để xóa danh sách giá ra khỏi hàng đợi clq

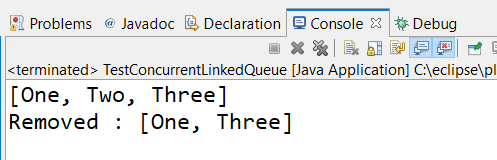
MyConcurrentLinkedQueue.*removeIf*(clq, "Two");

// Hiển thị hàng đợi sau khi xóa

System.***out***.println("Removed : "+ clq);

}

- Kết quả chạy



3.12 Kịch bản chạy cho phương thức spliterator()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

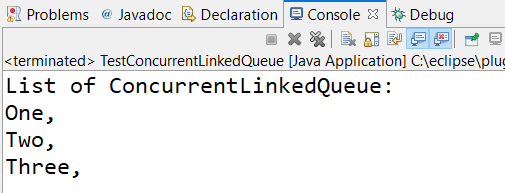
MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// Sử dụng Spliterator để in hàng đợi ra màn hình

MyConcurrentLinkedQueue.*printQueueBySpliterator*(clq);

}

- Kết quả chạy



3.13 Kịch bản chạy cho phương thức toArray()

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// Hiển thị hàng đợi hiện có

System.***out***.println("ConcurrentLinkedQueue: "+clq);

//Chuyển hàng đợi thành mảng một chiều

Object[] strArray = MyConcurrentLinkedQueue.*toArray*(clq);

// Hiển thị mảng một chiều

System.***out***.println("Array: ");

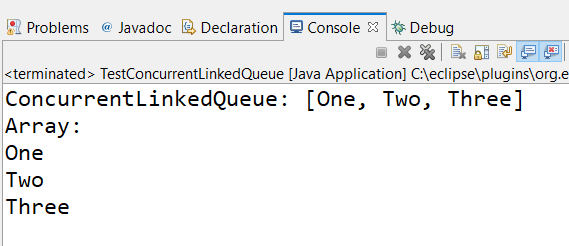
**for** (Object str : strArray) {

System.***out***.println(str);

}

}

- Kết quả chạy



3.14 Kịch bản chạy cho phương thức lấy phần tử trong ConcurrentLinkedQueue

- Kịch bản chạy

**public** **static** **void** main(String[] args) {

// Tạo một ConcurrentLinkedQueue

ConcurrentLinkedQueue<String> clq =

MyConcurrentLinkedQueue.*createConcurrentLinkedQueue*();

// Hiển thị hàng đợi hiện có

System.***out***.println("ConcurrentLinkedQueue: " + clq);

String value = "";

// Sử dung peek()

value = MyConcurrentLinkedQueue.*getValueHeadByPeek*(clq);

System.***out***.println("peek() = "+value);

System.***out***.println("ConcurrentLinkedQueue after use peek(): " +clq);

// Sử dung element()

value = MyConcurrentLinkedQueue.*getValueHeadByElement*(clq);

System.***out***.println("element() = "+value);

System.***out***.println("ConcurrentLinkedQueue after use element(): " +clq);

// Sử dung peek()

value = MyConcurrentLinkedQueue.*getValueHeadByPoll*(clq);

System.***out***.println("poll() = "+value);

System.***out***.println("ConcurrentLinkedQueue after use poll() : " +clq);

}

- Kết quả chạy

