Chương 2: ArrayBlockingQueue

2.1 Tổng quan về ArrayBlockingQueue

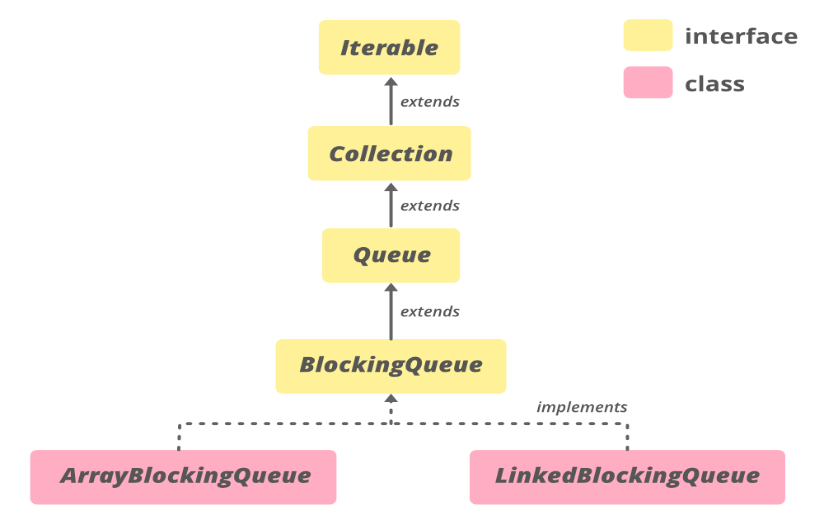
ArrayBlockingQueue là một hàng đợi có kích thước cố định dưới dạng mảng. Kích thước hàng đợi sẽ được cố định, đồng nghĩa là không thể thay đổi sau khi khởi tạo.

Nếu cố thêm dữ liệu vào hàng đợi, điều đó sẽ ngăn cản sự hoạt động của nó. Và tương tự với việc lấy dữ liệu từ một hàng đợi rỗng thì cũng sẽ bị chặn.

Sự rằng buộc của ArrayBlockingQueue có thể bỏ qua lưu trữ ban đầu thông qua tham số được truyền tại constructor trong ArrayBlockingQueue. Hiểu đơn giản là yêu cầu đầu vào của ArrayBlockingQueue sẽ được truyền qua tham số. ArrayBlockingQueue có dạng FIFO (first – in – first - out). Dữ liệu được thêm vào sẽ được đẩy về cuối hàng đợi và hàng đợi sẽ thực thi các giá trị phía đầu hàng.

ArrayBlockingQueue được thực hiện tất cả các phương thức của Collection và Iterator interface.

*Interface* là một cách khai báo phương thức, nhưng không chứa logic. Các class khác sẽ kế thừa interface và dùng các phương thức được interface khai báo sẵn. Interface có thể implement nhiều interface.

*Implement* là một các hướng đa kế thừa được java cung cấp. Một class chỉ có thể extends một class khác. Trong khi đó, một class có thể implement nhiều interface. Đồng nghĩa với việc, sẽ mở rộng dễ dàng hơn. Giảm tính phụ thuộc giữa các phương thức. Vì implement chỉ kế thừa phương thức, chứ không kế thừa logic, vì vậy implement có thể tái sử dụng nhiều lần mà không bị conflict code logic.

*Hình 2.1: Mô hình phân cấp lớp hệ thống của ArrayBlockingQueue*

2.2 Hàm khởi tạo của ArrayBlockingQueue

**public class** ArrayBlockingQueue<E> **extends** AbstractQueue<E> **implements** BlockingQueue<E>, Serializable

2.3 Hàm khởi tạo của ArrayBlockingQueue

* + 1. **ArrayBlockingQueue(int capacity)**

**Tạo một ArrayBlockingQueue với kích thước được cố địch từ ban đầu (capacity) và cách thức truy cập mặc định.**

**Cú pháp:**

**// Khai báo đối tượng hàng đợi kiểu Interger**

**int capacity = 10;**

**ArrayBlockingQueue<Integer> adp = new ArrayBlockingQueue<>(capacity)**

* + 1. **ArrayBlockingQueue(int capacity, boolean fair)**

**Tạo một ArrayBlockingQueue với kích thước được cố địch từ ban đầu và cách thức truy cập mặc định. Nếu giá trị của *fair* là true thì truy cập hàng đợi cho các chuỗi bị chặn khi chèn hoặc loại bỏ, được xử lý theo thứ tự FIFO (First In First Out). Nếu là *false* thì thứ tự truy cập là không xác định.**

**Cú pháp:**

**// Khai báo đối tượng hàng đợi kiểu Interger**

**int capacity = 10;**

**Boolean fair = true;**

**ArrayBlockingQueue<Integer> adp = new ArrayBlockingQueue<>(capacity, fair)**

***Hình:***

* + 1. **ArrayBlockingQueue(int capacity, boolean fair, Collection c)**

**Tạo một ArrayBlockingQueue với kích thước được cố địch từ ban đầu và cách thức truy cập mặc định chứa các phần tử được lấy từ biến *c*, được thêm vào theo thứ tự duyệt của trình lặp của tập hợp. Nếu giá trị của *fair* là true thì truy cập hàng đợi cho các chuỗi bị chặn khi chèn hoặc loại bỏ, được xử lý theo thứ tự FIFO (First In First Out). Nếu là *false* thì thứ tự truy cập là không xác định**

**Cú pháp:**

**// Khai báo đối tượng hàng đợi kiểu Interger**

**int capacity = 10;**

**Boolean fair = true;**

**Collection c = new Collection();**

**ArrayBlockingQueue<Integer> adp = new DFSDFSDF ArrayBlockingQueue<>(capacity, fair, c)**

* 1. Phương thức cơ bản

2.4.1 Thêm dữ liệu

*add(E e)* là phương thức thêm phần tử được truyền qua tham số vào hàng đợi đang thực thi phương thức này. Nếu việc thêm phần tử mới vào hàng đợi vượt quá giới hạn lưu trữ hiện có của hàng đợi thì sẽ đưa ra lỗi **IllegalStateException**. Phương thức sẽ trả về **true** nếu thêm thành công phần tử, ngược lại sẽ đưa ra lỗi **IllegalStateException.**

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

{

        // Định nghĩa kích thước của hàng đợi

**int** capacity = 15;

        // Tạo đối tượng hàng đợi

        ArrayBlockingQueue<Integer> abq = **new** ArrayBlockingQueue<Integer>(capacity);

        // Thêm giá trị

        abq.add(1);

        abq.add(2);

        abq.add(3);

        // Hiển thị hàng đợi

        System.**out**.println("ArrayBlockingQueue:" + abq);

 }

Giá trị hiển thị

ArrayBlockingQueue:[1, 2, 3]

* + 1. Loại bỏ dữ liệu

*remove(Objec o)* là phương thức loại bỏ một phần tử hoặc một thực thể của chính hàng đợi đang gọi tới nó. Nếu hàm trả về *True* đồng nghĩa với việc là một phần tử hoặc một thực thể đã bị loại bỏ khỏi danh sách, ngược lại trả về *False* là không có phần tử nào bị loại bỏ.

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

     {

         // Định nghĩa kích thước của hàng đợi

**int** capacity = 15;

         // Tạo đối tượng hàng đợi

ArrayBlockingQueue<Integer> abq = **new** ArrayBlockingQueue<Integer>(capacity);

         // Thêm giá trị

         abq.add(1);

         abq.add(2);

         abq.add(3);

         // Hiển thị hàng đợi

         System.**out**.println("ArrayBlockingQueue:" + abq);

         // Loại bỏ giá trị 2

**boolean** res = abq.remove(2);

         // Hiển thị hàng đợi

         System.**out**.println("Loại bỏ giá trị 2 :" + res);

         // Hiển thị hàng đợi

         System.**out**.println("ArrayBlockingQueue:" + abq);

         // Loại bỏ tất cả giá trị của hàng đợi

         abq.clear();

         // Hiển thị lại hàng đợi

         System.**out**.println("ArrayBlockingQueue:" + abq);

}

Giá trị hiển thị

ArrayBlockingQueue:[1, 2, 3]

Loại bỏ giá trị 2: true

ArrayBlockingQueue:[1, 3]

ArrayBlockingQueue:[]

* + 1. Truy cập phần tử

*peek()* là phương thức được cung cấp bởi **Queue** interface, sử dụng để trả về giá trị đầu tiên của hàng đợi, nếu không có thì sẽ trả về giá trị *null.* Phương thức này chỉ truy cập, không làm loại bỏ phần tử.

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

{

// Định nghĩa kích thước của hàng đợi

**int** capacity = 5;

// Tạo đối tượng hàng đợi

ArrayBlockingQueue<Integer> queue = **new** ArrayBlockingQueue<Integer>(capacity);

// Thêm giá trị vào hàng đợi

queue.add(23);

queue.add(32);

queue.add(45);

queue.add(12);

// Hiển thị hàng đợi

System.**out**.println("ArrayBlockingQueue:" + queue);

// Hiển thị giá trị vị trí đầu của hàng đợi

// Bằng cách sử dụng peek()

System.**out**.println("Head of queue " + queue.peek());

}

Giá trị hiển thị

ArrayBlockingQueue:[23, 32, 45, 12]

Head of queue: 23

* + 1. Duyệt trình tự

*iterator()* là phương thức của lớp **ArrayBlockingQueue**, được sử dụng để trả về một trình tự lắp của các phần tử giống như hàng đợi (theo một trình tự nhất định). Các phần tử được trả về từ phương thức này chứa các phần tử theo thứ tự từ đầu đến cuối của hàng đợi. Sẽ trả về thông báo lỗi, nếu hàng đợi rỗng.

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

{

// Định nghĩa kích thước của hàng đợi

**int** capacity = 5;

// Tạo đối tượng hàng đợi

ArrayBlockingQueue<String> queue = **new** ArrayBlockingQueue<String>(capacity);

// Thêm giá trị vào hàng đợi

queue.offer(“Nhan Vien”);

queue.offer(“Giam Doc”);

queue.offer(“Bao Ve”);

queue.offer(“Nhan Su”);

// Hiển thị hàng đợi

System.**out**.println("ArrayBlockingQueue:" + queue);

// Gọi hàm iterator() và tạo iterator

Iterator interatorValue = queue.iterator();

System.**out**.println("\nDanh sách giá trị:");

**while** (iteratorValues.hasNext()) {

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

         }

}

Giá trị hiển thị

ArrayBlockingQueue:[Nhan Vien, Giam Doc, Bao Ve, Nhan Su]

Danh sách giá trị:

Nhan Vien

Giam Doc

Bao Ve

Nhan Su

* 1. Các phương thức của ArrayBlockingQueue

2.5.1 Phương thức *add(E e)*

Phương thức dùng để thêm phần tử vào cuối hàng đợi thông qua tham số truyền vào hàm. Nếu thêm thành công sẽ trả về *True,* ngược lại sẽ trả vè thông báo lỗi *IllegalStateException.*

Cú pháp:

/\*\*

\* Phương thức sẽ thêm phần tử mới vào hàng đợi với vị trí là cuối cùng \* Trả về True nếu thêm thành công, ngược lại sẽ trả về thông báo lỗi<br>

\* Create by: BVHA (12/05/2021)

\*

\* @param <E>

\* @param abq

\* @param e - Phần tử để thêm vào hàng đợi

\* **@return**

\*/

**public** **static** <E> **boolean** addElement(ArrayBlockingQueue<E> abq, E e)

{

**return** abq.add(e);

}

2.5.2 Phương thức *addAll(Colllection<E> c)*

Phương thức dùng để thêm phần tử từ danh sách được truyền qua tham số vào cuối hàng đợi. Nếu thêm thành công sẽ trả về *true,* ngược lại sẽ trả vè thông báo lỗi *IllegalStateException.*

Cú pháp:

/\*\*

\* Phương thức thực dùng để thêm mới một danh sách phần tử vào hàng đợi.

\* Trả về <i>True</i> nếu thêm tất cả phần tử từ danh sách vào hàng đợi \* thành công. Ngược lại, trả về thông báo lỗi. <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** collec - Danh sách thêm vào hàng đợi

\* **@return**

\*/

**public** **static** <E> **boolean** addAllElement(ArrayBlockingQueue<E> abq, Collection<E> collec)

{

**return** abq.addAll(collec);

}

2.5.3 Phương thức *offer(E e)*

Phương thức thêm phần tử vào cuối hàng đợi được truyền qua tham số. Trả về *True* nếu thêm thành công, ngược lại trả về *False (hàng đợi đầy).* Phương thức này khác với phương thức *add,* bởi vì phương thức *add* sẽ thông báo lỗi khi hàng đợi đầy, còn phương thức *offer* sẽ trả về *false.*

Cú pháp:

/\*\*

\* Phương thức được dùng để thêm mới phần tử <b>e</b> vào hàng đợi với \* vị trí là cuối cùng Trả về <i>True</i> nếu thêm thành công, ngược lại \* sẽ trả về <i>False</i> <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq

\* **@param** e - Phần tử để thêm vào hàng đợi

\* **@return**

\*/

**public** **static** <E> **boolean** addElementByOffer(ArrayBlockingQueue<E> abq, E e)

{

**return** abq.offer(e);

}

2.5.4 Phương thức *offer(E e, long timeout, TimeUnit unit)*

Phương thức thêm phần tử vào cuối hàng đợi được truyền qua tham số. Nếu hàng đợi đầy, phương thức sẽ đợi một khoảng thời gian cho đến khi có khoảng trống trong hàng đợi. Trả về *true* nếu thêm thành công, ngượi lại sẽ thực hiện chờ đợi khoảng trống trong hàng đợi. Phương thức rất phù hợp khi chúng ta muốn đợi hàng đợi loại bỏ phần tử của nó, khi hàng đợi không đầy sẽ thực hiện thêm phần tử vào hàng đợi .

Cú pháp:

/\*\*

\* Phương thức được dùng để thêm mới phần tử <b>e</b> vào hàng đợi. Nếu \* hàng đợi đầy,thì hàm sẽ thực hiện đợi một khoảng thời gian <i>time</i>, \* với cách tính thời gian dựa vào <i>unit</i> Trả về <i>True</i> nếu \* thêm thành công, ngược lại trả về <i>False</i> <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** e - Phần tử để thêm vào hàng đợi

\* **@param** time - Thời gian chờ

\* **@param** unit - Đơn vị tính thời gian chờ

\* **@return**

\*/

**public** **static** <E> **boolean** addElementByOffer(ArrayBlockingQueue<E> abq, E e, **long** time, TimeUnit unit)

{

**try** {

**return** abq.offer(e, time, unit);

} **catch** (InterruptedException e1) {

e1.printStackTrace();

**return** **false**;

}

}

2.5.5 Phương thức *put(E e)*

Phương thức thêm phần tử vào cuối hàng đợi nếu hàng đợi không đầy, ngược lại sẽ đợi cho đến khi có thể thêm mới phần tử vào cuối hàng đợi.

Cú pháp:

/\*\*

\* Phương thức dùng để thêm phần tử mới vào hàng đợi. Nếu hàng đợi chưa \* đầy, thì thực hiện thêm mới phần tử vào cuối hàng đợi. Ngược lại, sẽ \* đợi cho đến khi có vị trí thêm mới. <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq

\* **@param** e

\* **@throws** InterruptedException

\*/

**public** **static** <E> **void** addElementByPut(ArrayBlockingQueue<E> abq, E e) **throws** InterruptedException

{

abq.put(e);

}

2.5.6 Phương thức *element()*

Phương thức trả về phần tử đầu tiên trong hàng đợi, không có tác dụng loại bỏ phần tử. Trả về lỗi *NoSuchElementException* nếu hàng đợi rỗng.

Cú pháp:

/\*\*

\* Trả về giá trị đầu tiên của hàng đợi nếu hàng đợi không rỗng. Ngược \* lại trả về thông báo lỗi <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@return**

\*/

**public** **static** <E> E getElementFirstByElement(ArrayBlockingQueue<E> abq) {

**return** abq.element();

}

2.5.7 Phương thức *peek()*

Phương thức trả về phần tử đầu tiên trong hàng đợi, không có tác dụng loại bỏ phần tử. Trả về *null* nếu hàng đợi rỗng.

Cú pháp:

/\*\*

\* Trả về giá trị đầu tiên của hàng đợi nếu hàng đợi không rỗng. Ngược \* lại trả về <i>null</i><br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@return**

\*/

**public** **static** <E> E getElementFirstByPeek(ArrayBlockingQueue<E> abq)

{

**return** abq.peek();

}

2.5.8 Phương thức *poll()*

Phương thức trả về và đồng thời loại bỏ phần tử đầu tiên của hàng đợi. Trả về *null* khi hàng đợi rỗng.

Cú pháp:

/\*\*

\* Phương trả về giá trị đầu tiên của hàng đợi, đồng thời loại bỏ giá \* trị đó khỏi hàng đợi <br>

\* Trả về <i>null</i> khi hàng đợi rỗng Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@return**

\*/

**public** **static** <E> E getAndRemoveFirstElement(ArrayBlockingQueue<E> abq) {

**return** abq.poll();

}

2.5.9 Phương thức *poll(long time, TimeUnit unit)*

Phương thức trả về và đồng thời loại bỏ phần tử đầu tiên của hàng đợi. Thực hiện đợi một khoảng thời gian (người lập trình cài đặt) cho đến khi có thể trả về và loại bỏ phần tử đầu tiên của hàng đợi

Cú pháp:

/\*\*

\* Phương thức trả về giá trị đầu tiên của hàng đợi. Thực hiện đợi khi \* hàng đợi rỗng <br> Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** time - Thời gian chờ

\* **@param** unit - Đơn vị tính thời gian chờ

\* **@return**

\* **@throws** InterruptedException

\*/

**public** **static** <E> E getAndRemoveFirstElement(ArrayBlockingQueue<E> abq, **long** time, TimeUnit unit) **throws** InterruptedException

{

**return** abq.poll(time, unit);

}

2.5.10 Phương thức *remove(), removeAll(), clear()*

- *remove(E e)*

Phương thức thực hiện loại bỏ phần tử trong hàng đợi theo giá trị của biến truyền vào. Trả về *true* nếu loại bỏ thành công, ngược lại trả về *false.*

Cú pháp:

/\*\*

\* Phương thức loại bỏ một phần tử theo giá trị <b>e</b> tồn tại trong \* hàng đợi, thứ tự từ FIFO. Trả về <i>True</i>nếu loại bỏ thành công, \* ngược lại trả về <i>False</i> <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợ

\* **@param** e - Phần tử loại bỏ khỏi hàng đợi

\* **@return**

\*/

**public** **static** <E> **boolean** removeElementByValue(ArrayBlockingQueue<E> abq, E e)

{

**return** abq.remove(e);

}

*- remove()*

Phương thức trả về phần tử đầu và đồng thời loại bỏ khỏi hàng đợi. Nếu hàng đợi rỗng, trả về NoSuchElementException.

Cú pháp:

/\*\*

\* Phương thức trả về phần tử đầu tiên, đồng thời loại bỏ phần tử đó khỏi \* hàng đợi. Trả về thông báo lỗi nếu hàng đợi rỗng.<br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@return**

\*/

**public** **static** <E> E removeElementFirst(ArrayBlockingQueue<E> abq)

{

**return** abq.remove();

}

*- removeAll(Collection c)*

Phương thức dùng để loại bỏ từ một tất cả phần tử của danh sách dựa vào phần tử của danh sách truyền qua tham số. Trả về *true* nếu hàm được thực thi, ngược lại trả về *NullPointerException* nếu danh sách truyền vào *null .*

Cú pháp:

/\*\*

\* Phương thức loại bỏ các phần tử dựa vào danh sách phần tử được truyền \* vào. Trả về <i>true</i> nếu hàm được thực thi, ngược lại trả về thông \* báo lỗi nếu danh sách được truyền vào là <i>null</i><br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq

\* **@param** c

\* **@return**

\*/

**public** **static** <E> **boolean** removeAllElement(ArrayBlockingQueue<E> abq, Collection<E> c)

{

**return** abq.removeAll(c);

}

*- removeIf()*

Phương thức dùng để loại bỏ phần tử của danh sách dựa vào hàm lọc điều kiện, nó được truyền qua tham biến của hàm. Trả về *true* nếu có khả năng loại bỏ phần tử, ngược lại trả về *NullPointerException* nếu hàng đợi *null.*

Cú pháp:

/\*\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@return**

\*/

**public** **static** <E> E removeElementFirst(ArrayBlockingQueue<E> abq)

{

**return** abq.removeIf(Predicate filter);

}

*- clear()*

Phương thức loại bỏ tất cả phần tử của hàng đợi. Sau khi thực thi, hàng đợi sẽ rỗng.

Cú pháp:

/\*\*

\* Phương thức loại bỏ tất cả các phần tử của hàng đợi. Hàng đợi sẽ rỗng \* sau khi gọi hàm này <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\*/

**public** **static** <E> **void** clear(ArrayBlockingQueue<E> abq) {

abq.clear();

}

2.5.11 Phương thức *contains(E e), containsAll(Collection c)*

*- contains()*

Phương thức kiểm tra tồn tại của phần tử trong tham số của hàm có trong hàng đợi hay không. Nếu tồn tại trả về *true,* ngược lại trả về *false.*

Cú pháp:

/\*\*

\* Phương thức kiểm tra tồn tại của phần tử <b>e</b> có trong hàng đợi. \* Trả về True nếu tồn tại, ngược lại trả về False <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** e - Phần tử để kiểm tra tồn tại

\* **@return**

\*/

**public** **static** <E> **boolean** containElement(ArrayBlockingQueue<E> abq, E e) {

**return** abq.contains(e);

}

*- containsAll()*

Phương thức dùng để kiểm tra xem hai danh sách có cùng chứa chung những phần tử hay không. Hàm sẽ nhận một danh sách truyền qua tham số và trả về *true* nếu tất cả phần tử của danh sách hiện tại tồn tại danh sách được truyền vào kia, ngược lại trả về *false.*

Cú pháp:

/\*\*

\* Phương thức kiểm tra tồn tại của tất cả phần tử trong <b>abq2</b> tồn \* tại trong <b>abq</b> Trả về <i>True</i> nếu tồn tại, ngược lại trả về

\* <i>False</i> <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** abq2 - Danh sách giá trị để kiểm tra tồn tại trong hàng đợi

\* **@return** boolean

\*/

**public** **static** <E> **boolean** containAllElement(ArrayBlockingQueue<E> abq, Collection<E> abq2)

{

**return** abq.containsAll(abq2);

}

2.5.12 Phương thức *drainTo(Collection c), drainTo(Collection c, int maxE)*

*- drainTo(Collection c)*

Phương thức loại bỏ tất cả phần tử từ hàng đợi và thêm chúng vào danh sách. Trả về số lượng phần tử của drain. Và sẽ báo lỗi trong các trường hợp sau:

+ *UnsupportedOperationException*: Nếu không được hỗ trợ bởi danh sách.

+ *ClassCastException:* Nếu lớp của một phần tử trong hàng đợi này ngăn cản được thêm vào bộ sưu tập được chỉ đinh.

+ *NullPointerException:* Nếu bộ sưu tập rỗng.

+ *IllegalArgumentException:* Nếu bộ sư tập chỉ định là hàng đợi hoặc một số thuộc tính của một phần tử của hàng đợi này ngăn việc thêm vào bộ sưu tập được chỉ định.

Cú pháp:

/\*\*

\* Phương thức chuyển đổi phần tử của hàng đợi thành tập các phần tử

\* (Collection). Trả về số lượng phần tử đã chuyển đổi. <br>

\* Create by: BVHA (12/05/2021)

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** collec - Danh sách để chứa giá trị chuyển đổi

\* **@return**

\*/

**public** **static** <E> **int** drainToCollection(ArrayBlockingQueue<E> abq, Collection<E> collec) {

**return** abq.drainTo(collec);

}

*- drainTo(Collection c, int maxE)*

Phương thức dùng để chuyển một lượng phần tử cố định tới bộ sưu tập. Sau khi chuyển đổi, hàng đợi chỉ còn tồn tại những phần tử chưa chuyển đổi. Còn trả về giá trị và báo lỗi thì giống với mục *drainTo(Collection c).*

Cú pháp:

/\*\*

\* Phương thức chuyển đổi phần tử của hàng đợi thành tập các phần tử

\* (Collection). Trả về số lượng phần tử đã chuyển đổi. Được giới hạn phần \* tử chuyển đổi bởi <b>maxElements.</b> Giá trị lấy tuân theo FIFO, nếu

\* maxElements nhở hơn kích thước sẽ lấy đúng số lượng giới hạn, ngượi \* lại nếu lớn hơn thì sẽ lấy toàn bộ <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** collec - Danh sách để chứa giá trị chuyển đổi

\* **@param** maxElements - Số lượng tối đa được chuyển đổi

\* **@return**

\* **@throws** IOException

\*/

**public** **static** <E> **int** drainToCollection(ArrayBlockingQueue<E> abq, Collection<E> collec, **int** maxElements) **throws** IOException

{

**return** abq.drainTo(collec, maxElements);

}

2.5.13 Phương thức *isEmpty()*

Phương thức dùng để kiểm trả và kiểm chứng nếu một danh sách là rỗng hoặc không tồn tại. Nó sẽ trả về *True* nếu danh sách rỗng, ngược lại trả về *False.*

Cú pháp:

/\*\*

\* Phương thức kiểm tra hàng đợi có tồn tại giá trị. Trả về <i>True</i> \* nếu tồn tại, ngược lại trả về <i>False</i> <br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@return**

\*/

**public** **static** <E> **boolean** isEmpty(ArrayBlockingQueue<E> abq)

{

**return** abq.isEmpty();

}

2.5.14 Phương thức *toArray(), toArray(T[] arr )*

*- toArray()*

Phương thức dùng để tạo một mảng chứa những giá trị giống với hàng đợi hiện tại. Về cơ bản, phương thức thực thi sao chép tất cả phần tử của hàng đợi sang một mảng mới. Có thể coi phương thức này là cầu nối giữa mảng và danh sách.

Cú pháp:

/\*\*

\* Phương thức trả về một mảng chứa tất cả các giá trị của hàng đợi<br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@return**

\*/

**public** **static** <E> Object toArray(ArrayBlockingQueue<E> abq)

{

Object[] array = abq.toArray();

**return** array;

}

*- toArray(T[] arr)*

Phương thức dùng để tạo một mảng chữa những giá trị giống với hàng đợi hiện tại. Phương thức sẽ thêm một điều kiện bổ sung. Kiểu mảng trả về giống với mảng chỉ định được truyền vào, nếu kích thước hàng đợi nhỏ hơn hoặc bằng với mảng chỉ định. Nếu không một mảng mới được cấp phát với kiểu giống với mảng được chỉ định và kích thước của mảng bằng với kích thước của hàng đợi. Phương thức này được coi như cầu nối giữa mảng và bộ sưu tập.

Cú pháp:

/\*\*

\* Phương thức trả về một mảng chứ tất cả các giá trị của hàng đợi. The

\* <i>arr</i> sẽ chứa tất cả các phần tử của hàng đợi, nếu mảng có kích \* thước lớn hơn hoặc bằng kích thước hàng đợi<br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq - Hàng đợi

\* **@param** arr - Mảng kiểu dữ liệu

\* **@return**

\*/

**public** **static** <E> Object toArray(ArrayBlockingQueue<E> abq, E[] arr)

{

// Tạo một mảng đối tượng, với giá trị lấy từ hàng đợi

Object[] array = abq.toArray(arr);

**return** array;

}

2.5.15 Phương thức *remainingEmpty()*

Phương thức trả về số lượng phần tử có thể được thêm vào hàng đợi mà không bị chặn. Có thể hiểu là giá trị trả về là kích thước tổng trừ đi số lượng phần tử hàng đợi đang chứa.

Cú pháp:

/\*\*

\* Phương thức trả về số lượng phần tử có thể lưu trữ<br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq

\* **@return**

\*/

**public** **static** <E> **int** getCapacityEmpty(ArrayBlockingQueue<E> abq)

{

**return** abq.remainingCapacity();

}

2.5.16 Phương thức *spliterator()*

Phương thức trả về *Spliterator* trên các phân tử của hàng đợi. *Spliterator* có thể được sử dụng với Streams trong Java 8. *Spliterator* có thể duyệt các phần tử riêng lẻ và hàng loạt.

Cú pháp:

/\*\*

\* Phương thức trả về Spliterator dựa vào các phần tử trong hàng đợi<br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq

\* **@return**

\*/

**public** **static** <E> Spliterator getSpliteratorList(ArrayBlockingQueue<E> abq )

{

Spliterator<E> spliterators = abq.spliterator();

**return** spliterators;

}

2.5.17 Phương thức *forEach()*

Phương thức thực thi duyệt qua từng phần tử của danh sách, hàng đợi.

Cú pháp:

/\*\*

\* Hiển thị lần lượt các phần tử trong hàng đợi<br>

\* Create by: BVHA (12/05/2021)

\*

\* **@param** <E>

\* **@param** abq

\*/

**public** **static** <E> **void** printArrayBlockingQueue(ArrayBlockingQueue<E> abq) {

abq.forEach(e -> System.***out***.println(e));

}

2.6 Kịch bản chạy chương trình

2.6.1 Kịch bản khởi tạo đối tượng

- *Kịch bản chạy*

**public** **static** **void** fn\_constructor() {

// Hàm khởi tạo đối tượng có kích thước = 5

ArrayBlockingQueue<String> abq = MyArrayBlockingQueue.*createArrayBlockingQueue*(5);

// Thêm phần tử

abq.add("A");

abq.add("B");

// Hàm khởi tạo đối tượng có kích thước = 5, true

ArrayBlockingQueue<String> abq1 = MyArrayBlockingQueue.*createArrayBlockingQueue*(5, **true**);

// Thêm phần tử

abq1.add("A");

abq1.add("B");

// Hàm khởi tạo đối tượng có kích thước = 5, true, clo

Collection<String> clo = **new** ArrayList<String>();

// Thêm phần tử

clo.add("A");

clo.add("B");

clo.add("C");

ArrayBlockingQueue<String> abq2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(5, **true**, clo);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1: " + abq);

System.***out***.println("Hàng đợi 2: " + abq1);

System.***out***.println("Hàng đợi 3: " + abq2);

}

*- Kết quả*

Hàng đợi 1: [A, B]

Hàng đợi 2: [A, B]

Hàng đợi 3: [A, B, C]

2.6.2 Kich bản thêm phần tử

**Phương thức *add(), addAll()***

*- Kịch bản chạy*

**public** **static** **void** fn\_add() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> abq = MyArrayBlockingQueue.*createArrayBlockingQueue*(5);

// Thêm phần tử mới vào hàng đợi

MyArrayBlockingQueue.*addElement*(abq, 4);

// Hiển thị hàng đợi

System.***out***.println("Add: " + abq);

// Khởi tạo và thêm giá trị vào danh sách

Collection<Integer> clo = **new** ArrayList<Integer>();

clo.add(1);

clo.add(3);

clo.add(2);

// Thêm danh sách vào hàng đợi

MyArrayBlockingQueue.*addAllElement*(abq, clo);

// Hiển thị hàng đợi

System.***out***.println("Add all: " + abq);

}

*- Kết quả*

Add: [4]

Add all: [4, 1, 3, 2]

**Phương thức *offer()***

*- Kịch bản chạy*

**public** **static** **void** fn\_addOffer() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Thêm phần tử vào hàng đợi

System.***out***.println("Thêm 423: " + MyArrayBlockingQueue.*addElementByOffer*(queue, 423));

System.***out***.println("Thêm 243: " + MyArrayBlockingQueue.*addElementByOffer*(queue, 243));

// Kiểm tra đầy hàng đợi

**if** (MyArrayBlockingQueue.*getCapacityEmpty*(queue) == 0) {

System.***out***.println("Hàng đợi đầy");

System.***out***.println("Hàng đợi tồn tại " + queue);

} **else** {

System.***out***.println("Hàng đợi chưa đầy");

System.***out***.println("Hàng đợi tồn tại" + queue);

}

// Cố gắng thêm phần tử vào

System.***out***.println("Thêm 123: " + MyArrayBlockingQueue.*addElementByOffer*(queue, 123));

System.***out***.println("Thêm 321: " + MyArrayBlockingQueue.*addElementByOffer*(queue, 321));

}

*- Kết quả*

Thêm 423: true

Thêm 243: true

Hàng đợi đầy

Hàng đợi tồn tại [423, 243]

Thêm 123: false

Thêm 321: false

*- Kịch bản chạy*

**public** **static** **void** fn\_addOfferV2() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

System.***out***.println("Thêm 423: " + MyArrayBlockingQueue.*addElementByOffer*(queue, 433, 10, TimeUnit.***SECONDS***));

System.***out***.println("Thêm 456: " + MyArrayBlockingQueue.*addElementByOffer*(queue, 456, 10, TimeUnit.***SECONDS***));

// Kiểm tra đầy hàng đợi

**if** (MyArrayBlockingQueue.*getCapacityEmpty*(queue) == 0) {

System.***out***.println("Hàng đợi đầy");

System.***out***.println("Hàng đợi tồn tại " + queue);

} **else** {

System.***out***.println("Hàng đợi chưa đầy");

System.***out***.println("Hàng đợi tồn tại" + queue);

}

// Tiếp tục thêm nhân viên

System.***out***.println("Thêm 123: " + MyArrayBlockingQueue.*addElementByOffer*(queue, 123, 5, TimeUnit.***SECONDS***));

}

*- Kết quả*

Thêm 423: true

Thêm 456: true

Hàng đợi đầy

Hàng đợi tồn tại [433, 456]

**Phương thức *put()***

*- Kịch bản chạy*

**public** **static** **void** fn\_addPut() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Thêm mới phần tử

MyArrayBlockingQueue.*addElementByPut*(queue, 223);

MyArrayBlockingQueue.*addElementByPut*(queue, 456);

// Hiển thị phần tử

System.***out***.println("Hàng đợi: " + queue);

}

*- Kết quả*

Hàng đợi: [223, 456]

2.6.3 Kịch bản lấy phần tử

**Phương thức *element()***

*- Kịch bản chạy*

**public** **static** **void** fn\_element() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi: " + queue);

System.***out***.println("Giá trị vị trí đầu hàng đợi 1: " + MyArrayBlockingQueue.*getElementFirstByElement*(queue));

// Hiển thị mảng rỗng

System.***out***.println("Hàng đợi 2: " + queue2);

System.***out***.println("Giá trị vị trí đầu hàng đợi 2: " + MyArrayBlockingQueue.*getElementFirstByElement*(queue2));

}

*- Kết quả*

Hàng đợi: [123, 456]

Giá trị vị trí đầu hàng đợi 1: 123

Hàng đợi 2: []

Exception: java.util.NoSuchElementException

**Phương thức *peek()***

*- Kịch bản chạy*

**public** **static** **void** fn\_peek() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1: " + queue);

System.***out***.println("Giá trị vị trí đầu hàng đợi 1: " + MyArrayBlockingQueue.*getElementFirstByPeek*(queue));

System.***out***.println("Hàng đợi 1: " + queue + "\n");

// Hiển thị mảng rỗng

System.***out***.println("Hàng đợi 2: " + queue2);

System.***out***.println("Giá trị vị trí đầu hàng đợi 2: " + MyArrayBlockingQueue.*getElementFirstByPeek*(queue2));

}

*- Kết quả*

Hàng đợi 1: [123, 456]

Giá trị vị trí đầu hàng đợi 1: 123

Hàng đợi 1: [123, 456]

Hàng đợi 2: []

Giá trị vị trí đầu hàng đợi 2: null

**Phương thức *poll()***

- *Kịch bản chạy*

**public** **static** **void** fn\_poll() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1: " + queue);

System.***out***.println("Giá trị vị trí đầu hàng đợi 1: " + MyArrayBlockingQueue.*getAndRemoveFirstElement*(queue));

System.***out***.println("Hàng đợi 1: " + queue + "\n");

// Hiển thị mảng rỗng

System.***out***.println("Hàng đợi 2: " + queue2);

System.***out***.println("Giá trị vị trí đầu hàng đợi 2: " + MyArrayBlockingQueue.*getAndRemoveFirstElement*(queue2));

}

*- Kết quả*

Hàng đợi 1: [123, 456]

Giá trị vị trí đầu hàng đợi 1: 123

Hàng đợi 1: [123, 456]

Hàng đợi 2: []

Giá trị vị trí đầu hàng đợi 2: null

2.6.4 Kịch bản loại bỏ phần tử

**Phương thức *remove()***

- Kịch bản chạy

**public** **static** **void** fn\_removeV2() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1: " + queue);

System.***out***.println("Loại bỏ giá trị hàng đợi 1: " + MyArrayBlockingQueue.*removeElementFirst*(queue));

System.***out***.println("Hàng đợi 1: " + queue + "\n");

// Hiển thị mảng rỗng

System.***out***.println("Hàng đợi 2: " + queue2);

System.***out***.println("Loại bỏ giá trị hàng đợi 2: " + MyArrayBlockingQueue.*removeElementFirst*(queue2));

}

*- Kết quả*

Hàng đợi 1: [123, 456]

Loại bỏ giá trị hàng đợi 1: 123

Hàng đợi 1: [456]

Hàng đợi 2: []

**Phương thức *removeAll()***

**-** Kịch bản chạy

**public** **static** **void** fn\_removeAll() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

queue2.add(123);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1: " + queue);

System.***out***.println("Loại bỏ giá trị hàng đợi 1: " + MyArrayBlockingQueue.*removeAllElement*(queue, queue2));

System.***out***.println("Hàng đợi 1: " + queue + "\n");

// Hiển thị mảng rỗng

System.***out***.println("Hàng đợi 2: " + queue2);

System.***out***.println("Loại bỏ giá trị hàng đợi 2: " + MyArrayBlockingQueue.*removeAllElement*(queue2, queue));

}

**-**Kết quả

Hàng đợi 1: [123, 456]

Loại bỏ giá trị hàng đợi 1: true

Hàng đợi 1: [456]

Hàng đợi 2: [123]

Loại bỏ giá trị hàng đợi 2: false

**Phương thức *clear()***

**-** Kịch bản chạy

**public** **static** **void** fn\_clear() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

queue2.add(123);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1: " + queue);

MyArrayBlockingQueue.*clear*(queue);

System.***out***.println("Hàng đợi 1: " + queue + "\n");

System.***out***.println("Hàng đợi 2: " + queue2);

MyArrayBlockingQueue.*clear*(queue2);

System.***out***.println("Hàng đợi 2: " + queue2);

}

- *Kết quả*

Hàng đợi 1: [123, 456]

Hàng đợi 1: []

Hàng đợi 2: [123]

Hàng đợi 2: []

2.6.5 Kịch bản kiểm tra tồn phần tử

**Phương thức *contains()***

*- Kịch bản chạy*

**public** **static** **void** fn\_contains() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1 tồn tại 123: " + MyArrayBlockingQueue.*containElement*(queue, 123));

// Hiển thị mảng rỗng

System.***out***.println("Hàng đợi 2 tồn tại 123: " + MyArrayBlockingQueue.*containElement*(queue2, 123));

}

*- Kết quả*

Hàng đợi 1 tồn tại 123: true

Hàng đợi 2 tồn tại 123: false

**Phương thức *containsAll()***

**-** *Kịch bản chạy*

**public** **static** **void** fn\_containsAll() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Khởi tạo danh sách và thêm phần tử

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

list.add(123);

list.add(456);

// Hiển thị hàng đợi

System.***out***.println("Hàng đợi 1 tồn tại 123: " + MyArrayBlockingQueue.*containAllElement*(queue, list));

// Hiển thị mảng rỗng

System.***out***.println("Hàng đợi 2 tồn tại 123: " + MyArrayBlockingQueue.*containAllElement*(queue2, list));

}

*- Kết quả*

Hàng đợi 1 tồn tại 123: true

Hàng đợi 2 tồn tại 123: false

2.6.6 Kịch bản chuyển đổi danh sách

**Phương thức *drainTo()***

**-** *Kích bản chạy*

**public** **static** **void** fn\_draintTo() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

System.***out***.println("Trước khi drainTo\n" + queue);

MyArrayBlockingQueue.*drainToCollection*(queue, queue2);

System.***out***.println("Sau khi drainTo\n" + queue);

System.***out***.println("queue2: " + queue2);

}

- *Kết quả*

Trước khi drainTo:

[123, 456]

Sau khi drainTo: []

queue2: [123, 456]

**-** *Kịch bản chạy*

**public** **static** **void** fn\_drainToV2() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

System.***out***.println("Trước khi drainTo\n" + queue);

**try** {

MyArrayBlockingQueue.*drainToCollection*(queue, queue2, 1);

} **catch** (IOException e) {

e.printStackTrace();

}

System.***out***.println("Sau khi drainTo\n" + queue);

System.***out***.println("queue2: " + queue2);

}

*- Kết quả*

Trước khi drainTo

[123, 456]

Sau khi drainTo

[456]

queue2: [123]

2.6.7 Kịch bản chuyển đổi mảng

**Phương thức *toArray()***

*- Kịch bản chạy*

**public** **static** **void** fn\_ToArray() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

Object[] array = MyArrayBlockingQueue.*toArray*(queue);

Object[] array2 = MyArrayBlockingQueue.*toArray*(queue2);

System.***out***.println("Mảng 1:");

MyArrayBlockingQueue.*printArray*(array);

System.***out***.println("\nMảng 2:");

MyArrayBlockingQueue.*printArray*(array2);

}

*- Kết quả*

Mảng 1:

123

456

Mảng 2:

*- Kịch bản chạy*

**public** **static** **void** fn\_ToArrayV2() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

// Tạo mảng

Integer[] array = **new** Integer[2];

// gọi hàm toArray(T[] a)

Object[] ReturnArray = MyArrayBlockingQueue.*toArray*(queue, array);

System.***out***.println("Hàng đợi: " + queue);

System.***out***.println("\nMảng chuyển từ toArray():");

MyArrayBlockingQueue.*printArray*(array);

System.***out***.println("\nMảng trả về từ toArray():");

MyArrayBlockingQueue.*printArray*(ReturnArray);

}

*- Kết quả*

Hàng đợi: [123, 456]

Mảng chuyển từ toArray():

123

456

Mảng trả về từ toArray():

123

456

2.6.7 Kịch bản chuyển đổi dạng spliterator

**Phương thức *spliterator()***

*- Kịch bản chạy*

**public** **static** **void** fn\_spliterator() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

Spliterator<Integer> number1 = MyArrayBlockingQueue.*getSpliteratorList*(queue);

Spliterator<Integer> number2 = MyArrayBlockingQueue.*getSpliteratorList*(queue2);

System.***out***.println("Kích thước của number1 : " + number1.estimateSize());

System.***out***.println("Danh sách number1:");

//Dùng hàm forEachRemaining

number1.forEachRemaining((n) -> System.***out***.println(n));

System.***out***.println("Kích thước của number2 : " + number1.estimateSize());

System.***out***.println("Danh sách number2:");

//Dùng hàm forEachRemaining

number2.forEachRemaining((n) -> System.***out***.println(n));

}

*- Kết quả*

Kích thước của number1 : 2

Danh sách number1:

123

456

Kích thước của number2 : 2

Danh sách number2:

2.6.8 Kịch bản kiểm tra hàng đợi rỗng

**Phương thức *isEmpty()***

**-** Kịch bản chạy

**public** **static** **void** fn\_IsEmpty() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

// Hiển thị

System.***out***.println("Hàng đợi 1: " + MyArrayBlockingQueue.*isEmpty*(queue));

System.***out***.println("Hàng đợi 2: " + MyArrayBlockingQueue.*isEmpty*(queue2));

}

*- Kết quả*

Hàng đợi 1: false

Hàng đợi 2: true

2.6.9 Kịch bản lấy kích thước chưa sử dụng

**Phương thức *remainingCapacity()***

- *Kịch bản chạy*

**public** **static** **void** fn\_remainingCapacity() {

// khởi tạo đối tượng

ArrayBlockingQueue<Integer> queue = TestArrayBlockingQueue.*fn\_arrayBlocking*();

ArrayBlockingQueue<Integer> queue2 = MyArrayBlockingQueue.*createArrayBlockingQueue*(2);

System.***out***.println("Hàng đợi 1: " + queue);

System.***out***.println("Kích thước có thể lưu trữ hàng đợi 1: " + MyArrayBlockingQueue.*getCapacityEmpty*(queue));

System.***out***.println("Hàng đợi 2: " + queue2);

System.***out***.println("Kích thước có thể lưu trữ hàng đợi 2: " + MyArrayBlockingQueue.*getCapacityEmpty*(queue2));

}

*- Kết quả*

Hàng đợi 1: [123, 456]

Kích thước có thể lưu trữ hàng đợi 1: 0

Hàng đợi 2: []

Kích thước có thể lưu trữ hàng đợi 2: 2

Tài liệu tham khảo:

1. <https://www.geeksforgeeks.org/arrayblockingqueue-class-in-java/>
2. <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html>
3. <https://cafedev.vn/tu-hoc-java-arrayblockingqueue-trong-java/>