Chương 1: Tổng quan

* 1. Khái niệm về AbstractQueue
     1. Lớp **AbstractQueue** trong Java là một phần của **Java Collection Framework** và implements **Collection** interface và lớp **AbstractCollection**.

Nó cung cấp các triển khai cơ bản của một số **Queue** operations.

Các triển khai trong lớp này thích hợp khi triển khai cơ sở không cho phép các phần tử rỗng.

Các phương thức add, remove và element dựa trên offer, poll và peek, respectively, nhưng đưa ra throw exeptions thay vì chỉ ra lỗi thông qua trả về false.

* 1. AbstractQueue trong java
     1. Cách khai báo trong AbstractQueue

**public** **abstract** **class** AbstractQueue<E> **extends** AbstractCollection<E> **implements** Queue<E> {

//methods ...

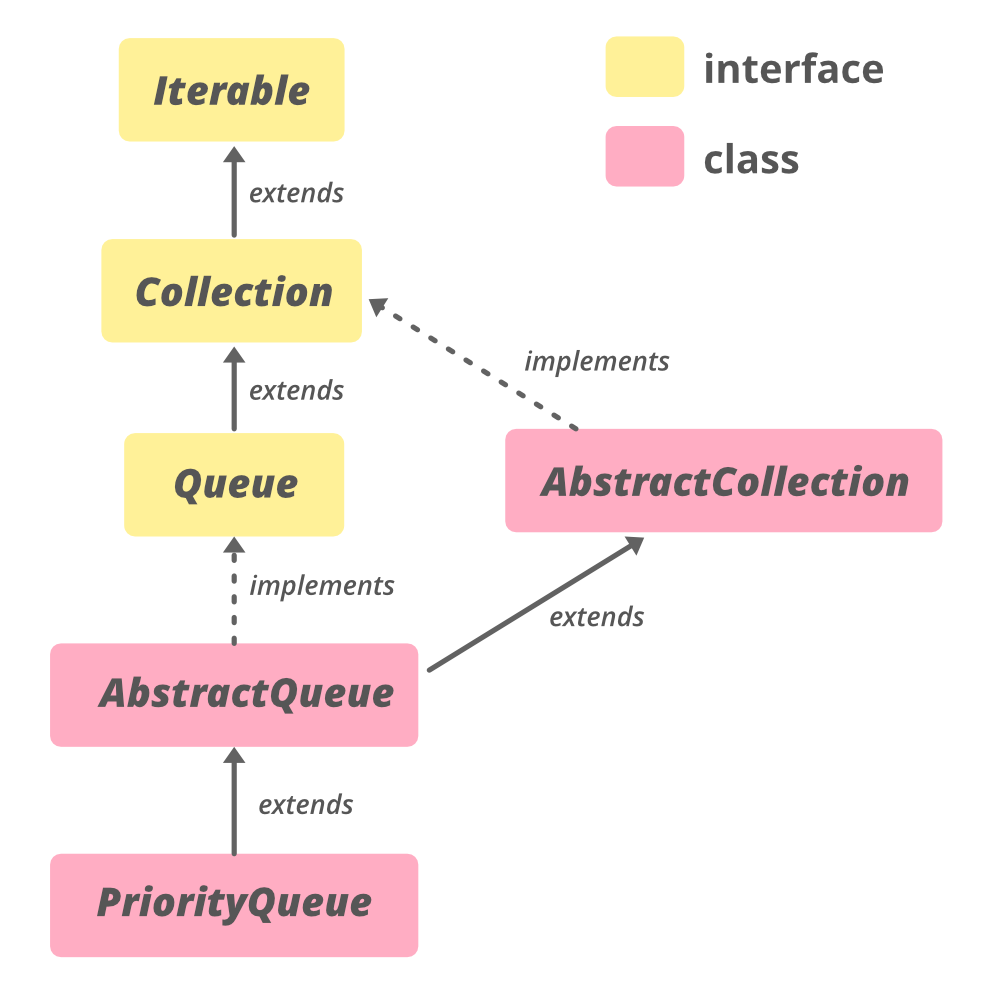
}

* + 1. Hệ thống phân cấp lớp

java.lang.Object

↳ java.util.AbstractCollection<E>

↳ Class AbstractQueue<E>



Lớp này implements Iterable<E>, Collection<E>, Queue<E> interfaces và

extends AbstractCollection.

* + 1. Cách tạo một AbstractQueue
* Vì AbstractQueue là một lớp trừu tượng, vì vậy việc định nghĩa nó được triển khai ở các sub-class của nó. Dưới đây là hiển thị danh sách các lớp con có thể cung cấp việc triển khai. Để tạo ra nó, chúng ta cần import thư viện **java.uti.AbstractQueue**.
* protected AbstractQueue(): hàm tạo mặc định, nhưng là trừu tượng, nó không cho phép tạo đối tượng AbstractQueue. Việc triển khai phải được cung cấp bởi một trong các lớp con của nó như là: [ArrayBlockingQueue](https://www.geeksforgeeks.org/arrayblockingqueue-class-in-java/#:~:text=ArrayBlockingQueue%20class%20is%20a%20bounded,result%20in%20the%20operation%20blocking.), [ConcurrentLinkedQueue](https://www.geeksforgeeks.org/concurrentlinkedqueue-in-java-with-examples/), [DelayQueue](https://www.geeksforgeeks.org/delayqueue-class-in-java-with-example/#:~:text=DelayQueue%20is%20a%20specialized%20Priority,queue%20whose%20time%20has%20expired.), [LinkedBlockingDeque](https://www.geeksforgeeks.org/linkedblockingdeque-in-java-with-examples/), [LinkedBlockingQueue](https://www.geeksforgeeks.org/linkedblockingqueue-class-in-java/#:~:text=LinkedBlockingQueue%20is%20an%20optionally%2Dbounded,the%20LinkedBlockingQueue%20will%20be%20unbounded.&text=It%20means%20that%20the%20head,elements%20present%20in%20this%20queue.), [LinkedTransferQueue](https://www.geeksforgeeks.org/linkedtransferqueue-in-java-with-examples/#:~:text=The%20LinkedTransferQueue%20class%20in%20Java,functionality%20based%20on%20linked%20nodes.), [PriorityBlockingQueue](https://www.geeksforgeeks.org/priorityblockingqueue-class-in-java/#:~:text=PriorityBlockingQueue%20is%20an%20unbounded%20blocking,resource%20exhaustion%20resulting%20in%20OutOfMemoryError.), [PriorityQueue](https://www.geeksforgeeks.org/priority-queue-class-in-java-2/), **SynchronousQueue**.

AbstractQueue<E> obj = new ArrayBlockingQueue<E>();

* + 1. Thêm một phần tử
* Để thêm các phần tử AbstractQueue, nó cung cấp 2 phương thức:
  + Phương thức add(E e) sẽ chèn phần tử được chỉ định vào hàng đợi này nếu có thể thực hiện điều đó ngay lập tức mà không vi phạm các hạn chế về dung lượng. Nó trả về true khi thành công và bật ra IllegalStateException nếu hiện không có dung lượng.
  + Phương thức addAll(E e) thêm tất cả các phần tử trong specified collection vào queue.

// Since AbstractQueue is an abstract class

// create object using LinkedBlockingQueue

AbstractQueue<E> AQ1 = **new**

LinkedBlockingQueue<E>();

// Populating AQ

AQ1.add(E e);

AQ1.add(E e1);

// Since AbstractQueue is an abstract class

// create object using LinkedBlockingQueue

AbstractQueue<E> AQ2 = **new**

LinkedBlockingQueue<E>();

// adds elements of AQ1 in AQ2

AQ2.addAll(AQ1);

* + 1. Xóa một phần tử
* Để loại bỏ các phần tử khỏi AbstractQueue, nó cung cấp các phương thức remove() và clear():
* Phương thức remove() trả về và loại bỏ phần tử đầu của queue này. Nếu chúng ta truyền vào giá trị cần remove vào phương thức thì phương thức sẽ check trong queue có tồn tại nó không và nếu có thì sẽ thực hiện xóa ngược lại thì queue sẽ vẫn giữ nguyên.
* Phương thức clear() xóa tất cả các phần tử khỏi queue này. Queue sẽ trống sau khi gọi hàm này.

// remove head element

AQ1.remove();

// remove element

AQ1.remove(E e);

// remove all elements

AQ2.clear();

* + 1. Truy cập một phần tử
* Phương thức element() của AbstractQueue truy xuất nhưng không loại bỏ, phần tử đầu tiên của queue.

AbstractQueue<E> AQ1 = **new**

LinkedBlockingQueue<E>();

AQ1.element();

Chương 2. Các phương thức của AbstractQueue

2.1 Phương thức add(E e)

- Phương thức add(E e) của AbstractQueue sẽ chèn thêm phần tử được chỉ định vào queue này nếu có thể làm như vậy ngay lập tức mà không vi phạm giới hạn dung lượng. Nó trả về true khi thành công và bật ra IllegaStateException nếu không có dung lượng.

- Cú pháp:

public boolean add(E e)

* Parameters: Phương thức chấp nhận một tham số bắt buộc e là phần tử được chèn vào queue.
* Returns: Phương thức này trả về true nếu phần tử được chèn vào queue ngược lại trả lại false.
* Exception: Phương thức này bật ra các Exception sau:

+ **IllegalStateException**: if the element cannot be added at this time due to capacity restrictions

+ **NullPointerException**: if the specified element is null

+ **ClassCastException** – if the class of the specified element prevents it from being added to this queue

+ **IllegalArgumentException** – if some property of this element prevents it from being added to this queue

/\*\*

\* Phương thức <b>thêm</b> một số int n vào queue. <br>

\* Phương thức này sẽ chèn số n vào cuối queue. <br>

\* Phương thức này sẽ trả về true nếu thêm thành công

\* ngược lại trả về false

\* **@param** AQ

\* **@param** n

\* **@return**

\*/

**public** **static** **boolean** add(AbstractQueue<Integer> AQ, **int** n)

{

**return** AQ.add(n);

}

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

Phương thức addAll () của AbstractQueue được sử dụng để chèn tất cả các phần tử của Collection, được truyền dưới dạng tham số cho phương thức này, vào cuối AbstractQueue. 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 số vào hàng đợi.</br>

\* - Phương thức này sẽ chèn một danh sách số 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** listInteger

\* **@return**

\*/

**public** **static** **boolean** addAll(AbstractQueue<Integer> queue,

ArrayList<Integer> listInteger) {

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

// danh sách vào cuối của hàng đợi.

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

}

2.2.2 Phương thức thêm từ AbstractQueue

/\*\*

\* - 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(AbstractQueue<Integer> queue,

AbstractQueue<Integer> 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 clear()

- Phương thức clear() của AbstractQueue loại bỏ tất cả các phần tử khỏi queue. Queue đó sẽ trống sau khi call returns.

/\*\*

\* - Phương thức xóa toàn bộ phần tử của AbstractQueue

\* **@param** AQ

\*/

**public** **static** **void** clear(AbstractQueue<Integer> AQ) {

// Phương thức clear() sẽ xóa tất cả các phần tử của AQ

AQ.clear();

}

2.4 Phương thức element()

- Phương thức element() của AbstractQueue truy xuất nhưng không loại bỏ phần tử đầu của Queue.

/\*\*

\* - Phương thức truy xuất phần tử đầu của AbstracQueue

\* **@param** AQ

\* **@return**

\*/

**public** **static** Integer element(AbstractQueue<Integer> AQ) {

**return** AQ.element();

}

2.5 Phương thức remove()

- Phương thức remove() của AbstractQueue loại bỏ phần tử có trong queue.

- Nếu không truyền tham số mặc định nó sẽ loại bỏ đi phần tử đầu của queue.

- Nếu truyền tham số thì nó sẽ loại bỏ đi phần tử tương đương với phần tử truyền vào trong queue nhưng nếu có nhiều phần tử trùng thì nó sẽ remove đi phần tử trùng với tham số truyền vào đầu tiên.

/\*\*

\* - Phương thức xóa 1 phần tử trong AbstractQueue

\* **@param** AQ

\* **@param** n

\* **@return**

\*/

**public** **static** **boolean** remove(AbstractQueue<Integer> AQ,

**int** n) {

**return** AQ.remove(n);

}

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[] argv) **throws** Exception {

// Since AbstractQueue is an abstract class

// create object using LinkedBlockingQueue

AbstractQueue<Integer> AQ1 = **new**

LinkedBlockingQueue<Integer>();

// Populating AQ

AQ1.add(10);

AQ1.add(20);

AQ1.add(30);

AQ1.add(40);

AQ1.add(50);

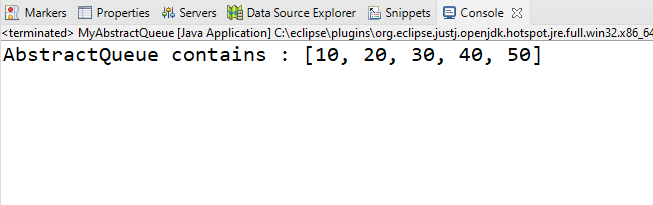
// print AQ

System.***out***.println("AbstractQueue contains : "

+ AQ1);

}

* Kết quả:



3.2 Kịch bản chạy phương thức add(E e)

- Kịch bản chạy:

**public** **static** **void** main(String[] argv) **throws** Exception {

// Since AbstractQueue is an abstract class

// create object using LinkedBlockingQueue

AbstractQueue<Integer> AQ1 = **new**

LinkedBlockingQueue<Integer>();

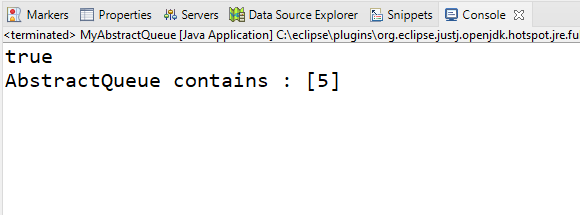
System.***out***.println(AQ1.add(5));

System.***out***.println("AbstractQueue contains : "

+ AQ1);

}

* Kết quả chạy:



3.2 Kịch bản chạy phương thức addAll()

- Kịch bản chạy:

**public** **static** **void** main(String[] argv) **throws** Exception {

// Since AbstractQueue is an abstract class

// create object using LinkedBlockingQueue

AbstractQueue<Integer> AQ1 = **new**

LinkedBlockingQueue<Integer>();

ArrayList<Integer> arrList = **new**

ArrayList<Integer>();

arrList.add(1);

arrList.add(2);

System.***out***.println(*addAll*(AQ1 ,arrList));

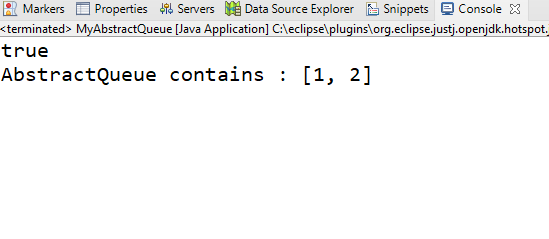
// print AQ

System.***out***.println("AbstractQueue contains : "

+ AQ1);

}

* Kết quả chạy:



3.3 Kịch bản chạy phương thức element()

- Kịch bản chạy:

**public** **static** **void** main(String[] argv) **throws** Exception

{

// Since AbstractQueue is an abstract class

// create object using LinkedBlockingQueue

AbstractQueue<Integer> AQ1 = **new**

LinkedBlockingQueue<Integer>();

// Populating AQ

AQ1.add(10);

AQ1.add(20);

AQ1.add(30);

AQ1.add(40);

AQ1.add(50);

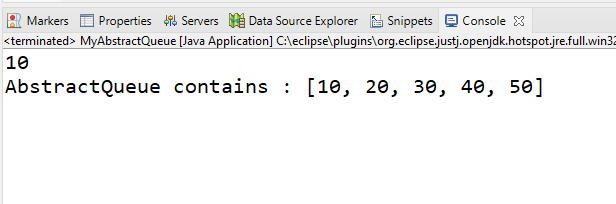
System.***out***.println(*element*(AQ1));

System.***out***.println("AbstractQueue contains : " +

AQ1;

}

* Kết quả:



3.4 Kịch bản chạy phương thức remove()

**public** **static** **void** main(String[] argv) **throws** Exception {

// Since AbstractQueue is an abstract class

// create object using LinkedBlockingQueue

AbstractQueue<Integer> AQ1 = **new**

LinkedBlockingQueue<Integer>();

// Populating AQ

AQ1.add(10);

AQ1.add(20);

AQ1.add(30);

AQ1.add(40);

AQ1.add(50);

System.***out***.println("AbstractQueue contains : " + AQ1);

System.***out***.println("Remove status: " + *remove*(AQ1, 20));

System.***out***.println("AbstractQueue contains : " + AQ1);

}

* Kết quả chạy:

