**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**======\*\*\*======**

****

**BÁO CÁO BTL THUỘC HỌC PHẦN:**

**LẬP TRÌNH JAVA**

**Đề tài:**

**Nghiên cứu các kỹ thuật xử lý của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque**

|  |  |
| --- | --- |
| GVHD: | Ths Hoàng Quang Huy |
| Nhóm - Lớp: | 10 – KTPM2K12 |
| Thành viên: | Phạm Tuấn Sơn - 2017604974 |
|  | Trần Lê Thịnh - 2017606165 |
|  | Vũ Đức Vượng - 2017604374 |

Hà nội, Năm 2020

**Lời nói đầu**

**Công nghệ thông tin** đã tạo nên một xã hội toàn cầu, nơi mà mọi người có thể tương tác và liên lạc với nhau một cách nhanh chóng và hiệu quả. Trong hơn 15 năm qua, lĩnh vực công nghệ thông tin Việt Nam đã đạt được nhiều thành tựu quan trọng.

Ngày nay, công nghệ thông tin dần trở thành ngành kinh tế mũi nhọn của đất nước, có tỉ lệ đóng góp cho tăng trưởng GDP đất nước ngày càng cao. Trải qua nhiều thời kì, các ngôn ngữ máy tính hay ngôn ngữ lập trình (Programming Language) ngày một ra đời. Và khi nhắc đến các ngôn ngữ lập trình thì không thể không nhắc đến Java. Java (một ngôn ngữ lập trình được phát triển bởi **Sun Microsystem** vào năm 1995, là ngôn ngữ kế thừa trực tiếp từ C/C++) là một ngôn ngữ nằm trong top đầu được sử dụng rộng rãi. Java là ngôn ngữ dễ học, nó lược bớt những lệnh thừa và rườm rà của C/C++ để người lập trình chú trọng vào việc viết chương trình. Đồng thời, nó còn hạn chế người lập trình không can thiệp quá sâu vào hệ thống.

Ngày nay, khi nhắc đến Java người ta không còn chỉ nhắc đến Java như là một ngôn ngữ mà nhắc đến Java như là một công nghệ hay một nền tảng phát triển.

Hàng đợi (Queue) là một khái niệm quan trọng trong Java, các kỹ thuật xử lý với hàng đợi trong Java không thể không nhắc đến đó là ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque là những kỹ thuật mà lấy đối tượng ở đầu hang đợi ra khỏi hàng đợi và trả về giá trị của nó. Nếu hàng đợi rỗng thì lỗi sẽ xảy ra.

Nhóm 10

**Lời cảm ơn**

Chúng em cảm ơn thầy đã giao cho chúng em đề tài “*Nghiên cứu các kỹ thuật xử lý của ArrayDeque, ConcurrentLinkDeque, LinkBlockingDeque*”. Thời gian nghiên cứu đề tài đã giúp chúng em hiểu sâu hơn về ngôn ngữ lập trình Java. Giúp chúng em cóp nhặt được niềm đam mê với ngôn ngữ Java nói riêng, và thế giới lập trình nói chung. Trong quá trình làm bài tập lớn môn học, do còn nhiều hạn chế về thời gian và kinh nghiệm thực tế, nhóm chúng em không tránh khỏi những thiếu sót mong nhận được những góp ý chân thành từ thầy và các bạn.

Chúng em trân thành cảm ơn.

**- Nhóm 10 -**

**Tóm tắt nội dung**

Bài tập lớn được chia làm 3 phần chính với 3 chương:

* Chương 1: Tìm hiểu về các kỹ thuật xử lý của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque
* Chương 2: Nghiên cứu các kỹ thuật xử lý của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque và viết các phương thức
* Chương 3: Kết quả trả về khi chạy các phương thức của các kỹ thuật xử lý của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque

MỤC LỤC

[Chương 1. Tìm hiểu về các kỹ thuật xử lý của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque 8](#_Toc42293034)

[1.1 Kỹ thuật xử lý của ArrayDeque trong Java 8](#_Toc42293035)

[1.2 Kỹ thuật xử lý của ConcurrentLinkedDeque trong Java 10](#_Toc42293036)

[1.3 LinkedBlockingDeque in Java 12](#_Toc42293037)

[- Phân cấp lớp : 12](#_Toc42293038)

[- Cú pháp : 12](#_Toc42293039)

[Chương 2. Nghiên Cứu Các Kỹ Thuật Xử Lý Của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque, Viết Phương Thức API 16](#_Toc42293040)

[2.1 Nghiên cứu về kỹ thuật xử lý của ArrayDeque trong Java 16](#_Toc42293041)

[2.1.1 Phương thức intializingIntegerDeque(): 16](#_Toc42293042)

[2.1.2 Phương thức addDeque(): 17](#_Toc42293043)

[2.1.3 Phương thức showDeque(): 17](#_Toc42293044)

[2.1.4 Phương thức addFirstDeque(): 17](#_Toc42293045)

[2.1.5 Phương thức addLastDeque(): 18](#_Toc42293046)

[2.1.6 Phương thức deepCopyDeque(): 18](#_Toc42293047)

[2.1.7 Phương thức clearDeque(): 19](#_Toc42293048)

[2.1.8 Phương thức sizeDeque(): 19](#_Toc42293049)

[2.1.9 Phương thức containsDeque(): 19](#_Toc42293050)

[2.1.10 Phương thức iteratorDeque(): 20](#_Toc42293051)

[2.1.11 Phương thức descendingIteratorDeque(): 20](#_Toc42293052)

[2.1.12 Phương thức fetchHeadArrayDeque(): 21](#_Toc42293053)

[2.1.13 Phương thức getFirstElement(): 21](#_Toc42293054)

[2.1.14 Phương thức getLastElement(): 21](#_Toc42293055)

[2.1.15 Phương thức isEmpty(): 21](#_Toc42293056)

[2.1.16 Phương thức toArray(): 22](#_Toc42293057)

[2.2 Nghiên cứu về ConcurrentLinkedDeque trong Java 22](#_Toc42293058)

[2.2.1 Phương thức show(): 22](#_Toc42293059)

[2.2.2 Phương thức add(): 23](#_Toc42293060)

[2.2.3 Phương thức addFirst(): 23](#_Toc42293061)

[2.2.4 Phương thức addLast: 24](#_Toc42293062)

[2.2.5 Phương thức getFirst(): 24](#_Toc42293063)

[2.2.6 Phương thức getLast(): 24](#_Toc42293064)

[2.2.7 Phương thức offer(): 25](#_Toc42293065)

[2.2.8 Phương thức offerFirst(): 25](#_Toc42293066)

[2.2.9 Phương thức offerLast(): 25](#_Toc42293067)

[2.2.10 Phương thức peekFirst(): 26](#_Toc42293068)

[2.2.11 Phương thức peekLast 26](#_Toc42293069)

[2.3 Nghiên cứu về LinkedBlockingDeque trong Java 27](#_Toc42293070)

[2.3.1 Phương thức show(): 27](#_Toc42293071)

[2.3.2 Phương thức add(): 27](#_Toc42293072)

[2.3.3 Phương thức addFirst(): 28](#_Toc42293073)

[2.3.4 Phương thức addLast(): 28](#_Toc42293074)

[2.3.5 Phương thức clear(): 29](#_Toc42293075)

[2.3.6 Phương thức contains(): 29](#_Toc42293076)

[2.3.7 Phương thức getElement(): 30](#_Toc42293077)

[2.3.8 Phương thức getFirstElement(): 30](#_Toc42293078)

[2.3.9 Phương thức getLastElement(): 30](#_Toc42293079)

[2.3.10 Phương thức iterator(): 31](#_Toc42293080)

[2.3.11 Phương thức peek(): 31](#_Toc42293081)

[2.3.12 Phương thức remove(): 31](#_Toc42293082)

[2.3.13 Phương thức size(): 32](#_Toc42293083)

[2.3.14 Phương thức take(): 32](#_Toc42293084)

[2.3.15 Phương thức toString(): 32](#_Toc42293085)

[Chương 3. Chạy Demo Các Phương Thức API 33](#_Toc42293086)

[3.1 Chạy chương trình 33](#_Toc42293087)

[3.1.1 Kết quả chạy các method của ArrayDeque 33](#_Toc42293088)

[3.1.2 Kết quả chạy các method của ConcurrentLinkedDeque 35](#_Toc42293089)

[3.1.3 Kết quả chạy các method của LinkedBlockingDeque 37](#_Toc42293090)

# Tìm hiểu về các kỹ thuật xử lý của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque

## Kỹ thuật xử lý của ArrayDeque trong Java

* ArrayDeque trong Java cung cấp một cách để áp dụng mảng có thể thay đổi kích thước ngoài việc triển khai giao diện Deque. Nó còn được gọi là Array Double Ended Queue hoặc Array Deck. Đây là một loại mảng đặc biệt phát triển và cho phép người dùng thêm hoặc xóa một phần tử khỏi cả hai phía của hàng đợi.
* Một số tính năng quan trọng của ArrayDeque như sau:
  + Deques mảng không có hạn chế dung lượng và sự phát triển khi cần thiết để hỗ trợ sử dụng.
  + Chúng không an toàn cho luồng, điều đó có nghĩa là trong trường hợp không có đồng bộ hóa bên ngoài, ArrayDeque không hỗ trợ truy cập đồng thời bởi nhiều luồng.
  + Các phần tử Null bị cấm trong ArrayDeque.
  + Lớp ArrayDeque có khả năng nhanh hơn Stack khi được sử dụng như một ngăn xếp.
* **Declaration:**

public class ArrayDeque *Element*

extends AbstractCollection

implements Deque*Element*, Cloneable, Serializable

* Ở đây, Element đề cập đến thành phần có thể tham chiếu đến bất kỳ lớp nào, chẳng hạn như lớp Integer hoặc String.

**Constructors in ArrayDeque:**

* **ArrayDeque ():** Được sử dụng để tạo một ArrayDeque trống và theo mặc định giữ một dung lượng ban đầu để chứa 16 phần tử.
* **ArrayDeque (Collection c):** Được sử dụng để tạo ArrayDeque chứa tất cả các thành phần giống như của bộ sưu tập đã chỉ định.
* **ArrayDeque (int numofElements):** Được sử dụng để tạo một ArrayDeque trống và giữ khả năng chứa một số phần tử đã chỉ định.

**Các phương thức trong ArrayDeque:**

1. add (Element e): Phương thức chèn phần tử cụ thể vào cuối deque.
2. addFirst (Element e): Phương thức chèn phần tử cụ thể vào lúc bắt đầu deque.
3. addLast (Element e): Phương thức chèn phần tử cụ thể vào cuối deque. Nó tương tự để thêm phương thức ()
4. clear (): Phương thức loại bỏ tất cả các phần tử deque.
5. size (): Phương thức trả về không. của các yếu tố trong deque.
6. clone (): Phương thức sao chép deque.
7. contains() :Phương thức kiểm tra xem một deque có chứa phần tử hay không
8. Iterator (): Phương thức trả về một iterator trên deque.
9. desceinatingIterator (): Phương thức trả về một trình vòng lặp thứ tự ngược qua deque
10. element (): Phương thức trả về phần tử ở phần đầu của deque
11. getFirst (): Phương thức trả về phần tử đầu tiên của deque
12. getLast (): Phương thức trả về phần tử cuối cùng của deque
13. isEmpty (): Phương thức kiểm tra xem deque có trống hay không.
14. toArray (): Phương thức trả về mảng có các phần tử của deque.
15. Offer (Element e): Phương thức chèn phần tử vào cuối deque.
16. OfferFirst (Element e): Phương thức chèn phần tử ở phía trước deque.
17. OfferLast (Element e): Phương thức chèn phần tử vào cuối deque.
18. peek (): Phương thức trả về phần tử head mà không loại bỏ nó.
19. peekFirst (): Phương thức trả về phần tử đầu tiên mà không loại bỏ nó.
20. peekLast (): Phương thức trả về phần tử cuối cùng mà không loại bỏ nó.
21. poll (): Phương thức trả về phần tử head và cũng loại bỏ nó
22. pollFirst (): Phương thức trả về phần tử đầu tiên và cũng loại bỏ nó
23. pollLast (): Phương thức trả về phần tử cuối cùng và cũng loại bỏ nó
24. pop (): Phương thức bật ra một phần tử cho ngăn xếp được lặp lại bởi deque
25. push(Element e) Phương thức đẩy một phần tử lên ngăn xếp được lặp lại bởi deque
26. remove (): Phương thức trả về phần tử head và cũng loại bỏ nó
27. removeFirst (): Phương thức trả về phần tử đầu tiên và cũng loại bỏ nó
28. removeLast (): Phương thức trả về phần tử cuối cùng và cũng loại bỏ nó
29. removeFirstOccurrence (Obj): Phương thức loại bỏ phần tử nơi nó xuất hiện lần đầu tiên trong deque.
30. removeLastOccurrence (Obj): Phương thức loại bỏ phần tử nơi nó xuất hiện lần cuối trong deque.

## Kỹ thuật xử lý của ConcurrentLinkedDeque trong Java

* Lớp ConcurrencyLinkedDeque trong Java là một phần của Java Collection Framework và thực hiện Collection interface và AbstractCollection class LinkedList concurrently.
* Phân cấp lớp :

java.lang.Object

↳ java.util.AbstractCollection<E>

↳ Class ConcurrentLinkedDeque<E>

* **Syntax:**

public abstract class ConcurrentLinkedDeque<E>

extends AbstractCollection<E>

implements Deque<E>, Serializable

Where **E** is the type of elements maintained

by this collection.

* **ConcurrentLinkedDeque()**: This constructor is used to construct an empty deque.
* **ConcurrentLinkedDeque(Collection<E> c)**: This constructor is used to construct a deque with the elements of the Collection passed as the parameter.

**Các phương thức trong Java ConcurrencyLinkedDeque:**

1. add(E e): Phương thức này Chèn phần tử được chỉ định vào phần đuôi của deque này..
2. addAll(Collection c): Phương thức này Nối tất cả các phần tử trong bộ sưu tập đã chỉ định vào cuối deque này, theo thứ tự chúng được trả về bởi bộ lặp iterator của bộ sưu tập được chỉ định.
3. addFirst(E e): Phương thức này Chèn phần tử được chỉ định ở phía trước của deque này.
4. addLast(E e): Phương thức này Chèn phần tử được chỉ định vào cuối deque này.
5. clear():Phương thức này Loại bỏ tất cả các phần tử khỏi deque này.
6. contains(Object o): Phương thức này Trả về true nếu deque này chứa ít nhất một phần tử e sao cho o.equals (e).
7. descendingIterator():Phương thức này Trả về một trình vòng lặp qua các phần tử trong deque này theo thứ tự tuần tự ngược.
8. element():Phương thức này Lấy, nhưng không loại bỏ, phần đầu của hàng đợi được biểu thị bằng deque này (nói cách khác, phần tử đầu tiên của deque này).
9. getFirst():Phương thức này Lấy, nhưng không xóa, yếu tố đầu tiên của deque này.
10. getLast():Phương thức này Lấy, nhưng không loại bỏ, phần tử cuối cùng của deque này.
11. isEmpty (): Phương thức này Trả về true nếu bộ sưu tập này không chứa phần tử.
12. iterator (): Phương thức này Trả về một iterator qua các phần tử trong deque này theo trình tự thích hợp.
13. Offer (E e): Phương thức này Chèn phần tử được chỉ định vào phần đuôi của deque này.
14. OfferFirst (E e): Phương thức này Chèn phần tử được chỉ định ở phía trước của deque này.
15. OfferLast (E e): Phương thức này Chèn phần tử được chỉ định vào cuối deque này.
16. peek (): Phương thức này Lấy, nhưng không xóa,phần đầu của hàng đợi được đại diện bởi deque này (nói cách khác, phần tử đầu tiên của deque này) hoặc trả về null nếu deque này trống.
17. peekFirst (): Phương thức này Lấy, nhưng không loại bỏ, phần tử đầu tiên của deque này hoặc trả về null nếu deque này trống.
18. peekLast (): Phương thức này Lấy, nhưng không xóa,
19. phần tử cuối cùng của deque này, hoặc trả về null nếu deque này trống.
20. poll (): Phương thức này Lấy và loại bỏ phần đầu của hàng đợi được biểu thị bằng deque này (nói cách khác, phần tử đầu tiên của deque này) hoặc trả về null nếu deque này trống.
21. pollFirst (): Phương thức này Lấy và loại bỏ phần tử đầu tiên của deque này,
22. hoặc trả về null nếu deque này trống.
23. pollLast (): Phương thức này Lấy và loại bỏ phần tử cuối cùng của deque này hoặc trả về null nếu deque này trống.
24. pop (): Phương thức này Bỏ qua một phần tử từ ngăn xếp được biểu thị bằng deque này.
25. push (E e): Phương thức này Đẩy một phần tử lên ngăn xếp được biểu thị bằng deque này (nói cách khác,ở đầu của deque này) nếu có thể làm điều đó ngay lập tức mà không vi phạm các hạn chế về năng lực, trở lại thành công khi thành công và ném IllegalStateException nếu không có chỗ trống.
26. remove (): Phương thức này Lấy và xóa phần đầu của hàng đợi được biểu thị bằng deque này (nói cách khác,yếu tố đầu tiên của deque này).
27. remove (Object o): Phương thức này Loại bỏ phần tử đầu tiên e sao cho o.equals (), nếu một phần tử như vậy tồn tại trong deque này.
28. removeFirst (): Phương thức này Lấy và loại bỏ phần tử đầu tiên của deque này.
29. removeFirstOccurrence (Object o): Phương thức này Loại bỏ phần tử đầu tiên e sao cho o.equals (e),nếu một yếu tố như vậy tồn tại trong deque này.
30. removeLast (): Phương thức này Lấy và loại bỏ phần tử cuối cùng của deque này.
31. removeLastOccurrence (Object o): Phương thức này Loại bỏ phần tử cuối cùng e sao cho o.equals (e), nếu một phần tử như vậy tồn tại trong deque này.
32. size (): Phương thức này Trả về số lượng phần tử trong deque này.
33. toArray (): Phương thức này Trả về một mảng chứa tất cả các phần tử trong deque này, theo trình tự thích hợp (từ phần tử đầu tiên đến phần tử cuối cùng).
34. toArray (T [] a): Phương thức này Trả về một mảng chứa tất cả các phần tử trong deque này, theo trình tự thích hợp (từ phần tử đầu tiên đến phần tử cuối cùng); kiểu thời gian chạy của mảng trả về là kiểu đã chỉ định.

## LinkedBlockingDeque in Java

* Lớp LinkedBlockingDeque trong java là một phần của [Java Collection Framework](https://www.geeksforgeeks.org/collections-in-java-2/) và kế thừa từ Collection interface và lớp trừu tượng.
* Nó cũng thực hiện kỹ thuật BlockingDeque và cung cấp một chức năng giới hạn tùy chọn dựa trên các nút được liên kết. Giới hạn tùy chọn này được thể hiện bằng cách chuyển kích thước yêu cầu trong hàm tạo và giúp ngăn ngừa lãng phí bộ nhớ.
* Khi không được chỉ định, dung lượng theo mặc định được lấy là Integer.MAX\_VALUE.

# Phân cấp lớp :

java.lang.Object

↳ java.util.AbstractCollection<E>

↳ java.util.AbstractQueue<E>

↳ java.util.concurrent.LinkedBlockingDeque<E>

# Cú pháp :

public class LinkedBlockingDeque<E>

extends AbstractQueue<E>

implements BlockingDeque<E>, Serializable

Where **E** is the type of element maintained

by this collection.

* **LinkedBlockingDeque()**: This constructor is used to construct an empty deque. In this case the capacity is set to Integer.MAX\_VALUE
* **LinkedBlockingDeque(int capacity)**: This constructor creates a LinkedBlockingDeque with the given (fixed) capacity.
* **LinkedBlockingDeque(Collection<E> c)**: This constructor is used to construct a deque with the elements of the Collection passed as the parameter.

**Các phương thức trong Java LinkedBlockingDeque:**

1. add (E e): Phương thức này chèn phần tử được chỉ định vào cuối deque này trừ khi nó vi phạm các hạn chế về dung lượng.
2. addAll (Bộ sưu tập <E> c): Phương thức này nối thêm tất cả các phần tử trong bộ sưu tập đã chỉ định vào cuối deque này, theo thứ tự chúng được trả về bởi bộ lặp iterator của bộ sưu tập được chỉ định.
3. addFirst (E e): Phương thức này sẽ chèn phần tử được chỉ định ở phía trước của deque này nếu có thể làm điều đó ngay lập tức mà không vi phạm các hạn chế về dung lượng, ném IllegalStateException nếu không có chỗ trống.
4. addLast (E e): Phương thức này chèn phần tử được chỉ định vào cuối deque này nếu có thể làm điều đó ngay lập tức mà không vi phạm các hạn chế về dung lượng, ném IllegalStateException nếu không có chỗ trống.
5. clear (): Phương thức này loại bỏ tất cả các phần tử khỏi deque này.
6. contains(Object o): Phương thức này trả về true nếu deque này chứa phần tử đã chỉ định.
7. desceinatingIterator (): Phương thức này trả về một trình vòng lặp qua các phần tử trong deque này theo thứ tự tuần tự ngược.
8. DrainTo (Collection <E> c): Phương thức này loại bỏ tất cả các phần tử có sẵn khỏi hàng đợi này và thêm chúng vào bộ sưu tập đã cho.
9. DrainTo (Collection <E> c, int maxElements): Phương thức này loại bỏ tối đa số lượng phần tử có sẵn từ hàng đợi này và thêm chúng vào bộ sưu tập đã cho.
10. element(): Phương thức này lấy ra, nhưng không loại bỏ, phần đầu của hàng đợi được biểu thị bằng deque này.
11. forEach (Consumer<E> action): Phương thức này thực hiện hành động đã cho cho từng phần tử của Iterable cho đến khi tất cả các phần tử đã được xử lý hoặc hành động đưa ra một ngoại lệ.
12. getFirst (): Phương thức này lấy, nhưng không loại bỏ, phần tử đầu tiên của deque này.
13. getLast (): Phương thức này lấy, nhưng không loại bỏ, phần tử cuối cùng của deque này.
14. iterator (): Phương thức này trả về một iterator trên các phần tử trong deque này theo trình tự thích hợp.
15. Offer (E e): Phương thức này chèn phần tử được chỉ định vào hàng đợi được biểu thị bằng deque này (nói cách khác, ở phần đuôi của deque này) nếu có thể thực hiện ngay mà không vi phạm các hạn chế về dung lượng,

trả về đúng khi thành công và sai nếu không có chỗ trống hiện có.

1. Offer (E e, long timeout, TimeUnit unit): Phương thức này chèn phần tử được chỉ định vào hàng đợi được biểu thị bằng deque này (nói cách khác, ở phần đuôi của deque này), chờ đến thời gian chờ được chỉ định nếu cần cho không gian trở nên có sẵn.
2. OfferFirst (E e): Phương thức này chèn phần tử được chỉ định vào trước deque này nếu có thể làm điều đó ngay lập tức mà không vi phạm các hạn chế về năng lực, trả về đúng khi thành công và sai nếu không có chỗ trống.
3. OfferFirst (E e, long timeout, TimeUnit unit):Phương thức này chèn phần tử được chỉ định ở phía trước của deque này, chờ đến thời gian chờ đã chỉ định nếu cần thiết để không gian có sẵn.
4. OfferLast (E e): Phương thức này chèn phần tử được chỉ định vào cuối deque này nếu có thể làm điều đó ngay lập tức mà không vi phạm các hạn chế về dung lượng,trả về đúng khi thành công và sai nếu không có chỗ trống hiện có.
5. OfferLast(E e, long timeout, TimeUnit unit):Phương thức này chèn phần tử được chỉ định vào cuối deque này, chờ đến thời gian chờ đã chỉ định nếu cần để có chỗ trống.
6. peek (): Phương thức này lấy, nhưng không xóa,phần đầu của hàng đợi được đại diện bởi deque này (nói cách khác, phần tử đầu tiên của deque này) hoặc trả về null nếu deque này trống.
7. peekFirst (): Phương thức này lấy, nhưng không loại bỏ, phần tử đầu tiên của deque này hoặc trả về null nếu deque này trống.
8. peekLast (): Phương thức này lấy, nhưng không xóa,phần tử cuối cùng của deque này, hoặc trả về null nếu deque này trống.
9. poll (): Phương thức này lấy và loại bỏ phần đầu của hàng đợi được biểu thị bằng deque này (nói cách khác, phần tử đầu tiên của deque này) hoặc trả về null nếu deque này trống.
10. poll(long timeout, TimeUnit unit):Phương thức này truy xuất và loại bỏ phần đầu của hàng đợi được biểu thị bằng deque này (nói cách khác, phần tử đầu tiên của deque này), chờ đến thời gian chờ được chỉ định nếu cần thiết để một phần tử có sẵn.
11. pollFirst (): Phương thức này lấy và loại bỏ phần tử đầu tiên của deque này,hoặc trả về null nếu deque này trống.
12. pollFirst(long timeout, TimeUnit unit):Phương thức này lấy và loại bỏ phần tử đầu tiên của deque này, chờ đến thời gian chờ đã chỉ định nếu cần thiết để một phần tử có sẵn.
13. pollLast (): Phương thức này lấy và loại bỏ phần tử cuối cùng của deque này,hoặc trả về null nếu deque này trống.
14. pollLast(long timeout, TimeUnit unit):): Phương thức này lấy và loại bỏ phần tử cuối cùng của deque này, chờ đến thời gian chờ được chỉ định nếu cần thiết để một phần tử có sẵn.
15. pop (): Phương thức này bật một phần tử từ ngăn xếp được biểu thị bằng deque này.
16. push(E e): Phương pháp này đẩy một phần tử lên ngăn xếp được biểu thị bằng deque này (nói cách khác, ở đầu của deque này) nếu có thể làm điều đó ngay lập tức mà không vi phạm các hạn chế về năng lực, ném IllegalStateException nếu không có khoảng trống hiện đang có sẵn.
17. put (E e): Phương thức này chèn phần tử đã chỉ định vào hàng đợi được biểu thị bằng deque này (nói cách khác, ở phần đuôi của deque này), chờ đợi nếu cần để có khoảng trống.
18. putFirst (E e): Phương thức này chèn phần tử được chỉ định ở phía trước của deque này, chờ đợi nếu cần thiết để không gian có sẵn.
19. putLast (E e): Phương thức này chèn phần tử được chỉ định vào cuối deque này, chờ đợi nếu cần thiết để không gian có sẵn.
20. remainingCapacity():Phương thức này trả về số lượng phần tử bổ sung mà deque này có thể lý tưởng (trong trường hợp không có hạn chế về bộ nhớ hoặc tài nguyên) chấp nhận mà không chặn.
21. remove (): Phương thức này lấy và loại bỏ phần đầu của hàng đợi được biểu thị bằng deque này.
22. remove (Object o): Phương thức này loại bỏ sự xuất hiện đầu tiên của phần tử được chỉ định từ deque này.
23. remove ALL (Collection<E> c):Phương thức này loại bỏ tất cả các phần tử bộ sưu tập này cũng có trong bộ sưu tập được chỉ định (thao tác tùy chọn).
24. removeFirst (): Phương thức này lấy và loại bỏ phần tử đầu tiên của deque này.
25. removeFirstOccurrence (Object o): Phương thức này loại bỏ sự xuất hiện đầu tiên của phần tử được chỉ định từ deque này.
26. removeIf(Predicate<E> filter):Phương thức này loại bỏ tất cả các thành phần của bộ sưu tập này thỏa mãn vị từ đã cho.
27. removeLast (): Phương thức này lấy và loại bỏ phần tử cuối cùng của deque này.
28. removeLastOccurrence (Object o): Phương thức này loại bỏ sự xuất hiện cuối cùng của phần tử được chỉ định khỏi deque này.
29. retainAll(Collection<E> c):Phương thức này chỉ giữ lại các phần tử trong bộ sưu tập này có trong bộ sưu tập đã chỉ định (thao tác tùy chọn).
30. size (): Phương thức này trả về số lượng phần tử trong deque này.
31. spliterator (): Phương thức này trả về một Spliterator trên các phần tử trong deque này.
32. Take (): Phương thức này lấy và loại bỏ phần đầu của hàng đợi được biểu thị bằng deque này (nói cách khác, phần tử đầu tiên của deque này), chờ đợi nếu cần cho đến khi một phần tử có sẵn.
33. TakeFirst (): Phương thức này lấy và loại bỏ phần tử đầu tiên của deque này, chờ đợi nếu cần thiết cho đến khi một phần tử có sẵn.
34. TakeLast (): Phương thức này lấy và loại bỏ phần tử cuối cùng của deque này, chờ đợi nếu cần thiết cho đến khi một phần tử có sẵn.
35. toArray (): Phương thức này trả về một mảng chứa tất cả các phần tử trong deque này, theo trình tự thích hợp (từ phần tử đầu tiên đến phần tử cuối cùng).
36. toArray (T [] a): Phương thức này trả về một mảng chứa tất cả các phần tử trong deque này, theo đúng trình tự; kiểu thời gian chạy của mảng trả về là kiểu đã chỉ định.
37. toString (): Phương thức này trả về một chuỗi đại diện của bộ sưu tập này.

# Nghiên Cứu Các Kỹ Thuật Xử Lý Của ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque, Viết Phương Thức API

## Nghiên cứu về kỹ thuật xử lý của ArrayDeque trong Java

Cung cấp 1 cách để áp dụng mảng có thể thay đổi kích thước

Được phát triển để người dùng có thể thêm hoặc xoá một phần tử ở cả 2 phía của hàng đợi

Một số tính năng quan trọng:

* Không bị hạn chế và phát triển khi cần thiết để hỗ trợ sử dụng
* Không an toàn cho luồng, nên nếu không có đồng bộ hoá bên ngoài, mảng này không cho phép truy cập từ nhiều luồng.
* Phần tử Null bị cấm trong ArrayDeque.
* Có khả năng xử lý nhanh hơn Stack và LinkedList khi làm ngăn xếp và hàng đợi

Một số phương thức sử dụng với ArrayDeque:

### Phương thức intializingIntegerDeque():

* Chức năng: để khởi tạo một mảng ArrayDeque chứa các giá trị phần tử là số nguyên.
* Mã nguồn:

|  |
| --- |
| * *// intializing an integer Deque* * **public** **static** Deque<Integer> intializingIntegerDeque() { * *// Creating an empty ArrayDeque* * Deque<Integer> de\_que = **new** ArrayDeque<Integer>(); * **return** de\_que; * } |

### Phương thức addDeque():

* Chức năng: thêm một hoặc nhiều phần tử vào ArrayDeque do người dùng nhập từ bàn phím.
* Mã nguồn:

|  |
| --- |
| * **public** **static** Deque<Integer> addDeque(Deque<Integer> de\_que) { * *// Use add() method to add elements into the ArrayDeque* * System.***out***.println("Add value for the ArrayDeque (value = 000 to stop):"); * **while** (**true**) { * System.***out***.print("Add value: "); * **int** addValue = *sc*.nextInt(); * **if** (addValue == 000) * **break**; * de\_que.add(addValue); * } * *showDeque*(de\_que); * **return** de\_que; * } |

### Phương thức showDeque():

* Chức năng: Hiển thị ArrayDeque lên màn hình console.
* Mã nguồn:

|  |
| --- |
| * **public** **static** **void** showDeque(Deque<Integer> de\_que) { * System.***out***.println("ArrayDeque: " + de\_que); * System.***out***.println(""); * } |

### Phương thức addFirstDeque():

* Chức năng: thêm n (n do người dùng nhập) phần tử mới lên đầu ArrayDeque do người dùng nhập từ bàn phím.
* Mã nguồn:

|  |
| --- |
| **public** **static** Deque<Integer> addFirstDeque(Deque<Integer> de\_que) {   * *// n is the number of digits used for use with the addfirst () method* * System.***out***.print("Number of value to use with addFirst: "); * **int** n = *sc*.nextInt(); * *// Adding elements at the beggining* * System.***out***.println("Type value you want to use with addFirst(): "); * **for** (**int** i = 0; i < n; i++) { * de\_que.addFirst(*sc*.nextInt()); * } * **return** de\_que; } |

### Phương thức addLastDeque():

* Chức năng: thêm n (n do người dùng nhập) phần tử mới vào cuối ArrayDeque do người dùng nhập từ bàn phím.
* Mã nguồn:

|  |
| --- |
| * *// addLastDeque() method to add element to the end of ArrayDeque* * **public** **static** Deque<Integer> addLastDeque(Deque<Integer> de\_que) { * *// n is the number of digits used for use with the addfirst () method* * System.***out***.print("Number of value to use with addLast: "); * **int** n = *sc*.nextInt(); * *// Adding elements at the beggining* * System.***out***.println("Type value you want to use with addLast(): "); * **for** (**int** i = 0; i < n; i++) { * de\_que.addLast(*sc*.nextInt()); * } * **return** de\_que; * } |

### Phương thức deepCopyDeque():

* Chức năng: Tạo bản sao ArrayDeque với ô nhớ mới
* Mã nguồn:

|  |
| --- |
| * *// create a deep copy ArrayDeque (new memory)* * **public** **static** Deque<Integer> deepCopyDeQue(Deque<Integer> de\_que, Deque<Integer> copy\_de\_que) { * **for** (**int** value : de\_que) { * copy\_de\_que.add(value); * } * *showDeque*(copy\_de\_que); * **return** copy\_de\_que; * } |

### Phương thức clearDeque():

* Chức năng: xoá hết phần tử của ArrayDeque (nhưng không xoá ArrayDeque)
* Mã nguồn:

|  |
| --- |
| * */\** * *\* empty ArrayDeque / this method clears all element of array but does not* * *\* delete Deque* * *\*/* * **public** **static** Deque<Integer> clearDeQue(Deque<Integer> de\_que) { * *// Clearing the ArrayDeque* * de\_que.clear(); * *showDeque*(de\_que); * **return** de\_que; * } |

### Phương thức sizeDeque():

* Chức năng: Hiện thị độ dài của ArrayDeque
* Mã nguồn:

|  |
| --- |
| * *// show size of ArrayDeque* * **public** **static** **void** sizeDeque(Deque<Integer> de\_que) { * *// Displaying the size of ArrayDeque* * System.***out***.println("The size of de\_que is: " + de\_que.size()); * } |

### Phương thức containsDeque():

* Chức năng: Kiểm tra 1 giá trị có tồn tại trong ArrayDeque hay không.
* Mã nguồn:

|  |
| --- |
| * *// check or verify whether a specific element is present in the ArrayDeque or* * *// not* * **public** **static** **void** containsDeque(Deque<Integer> de\_que) { * System.***out***.print("check deque with value: "); * **int** x = *sc*.nextInt(); * System.***out***.println(de\_que.contains(x)); * System.***out***.println(""); * } |

### Phương thức iteratorDeque():

* Chức năng: Trả về 1 một vòng lặp các phần tử của ArrayDeque.
* Mã nguồn:

|  |
| --- |
| * *// return an iterator of the elements of the ArrayDeque.* * **public** **static** **void** iteratorDeque(Deque<Integer> de\_que) { * *// Creating an iterator* * Iterator<Integer> value = de\_que.iterator(); * *// Displaying the values after iterating through the ArrayDeque* * System.***out***.println("The iterator values are: "); * **while** (value.hasNext()) { * System.***out***.print(value.next() + "\t"); * } * System.***out***.println(""); * } |

### Phương thức descendingIteratorDeque():

* Chức năng: Trả về 1 vòng lặp (từ dưới lên) các phần tử của ArrayDeque.
* Mã nguồn:

|  |
| --- |
| * *// return an iterator of the elements of the ArrayDeque in descending order* * **public** **static** **void** descendingIteratorDeque(Deque<Integer> de\_que) { * *// Creating a desc\_iterator* * Iterator<Integer> value = de\_que.descendingIterator(); * *// Displaying the values after iterating through the ArrayDeque* * *// in reverse order* * System.***out***.println("The iterator values are: "); * **while** (value.hasNext()) { * System.***out***.print(value.next() + "\t"); * } * System.***out***.println(""); * } |

### Phương thức fetchHeadArrayDeque():

* Chức năng: Trả về phần tử đại diện ở đầu ArrayDeque.
* Mã nguồn:

|  |
| --- |
| *// retrieve or fetch the head of the ArrayDeque*  **public** **static** **void** fetchHeadArrayDeque(Deque<Integer> de\_que) {  *// Displaying the head*  System.***out***.println("The head element is: " + de\_que.element());  System.***out***.println("");  } |

### Phương thức getFirstElement():

* Chức năng: Lấy ra phần tử ở đầu ArrayDeque.
* Mã nguồn:

|  |
| --- |
| * *// get the first element of ArrayDeque* * **public** **static** **void** getFirstElement(Deque<Integer> de\_que) { * *// Displaying the first element* * System.***out***.println("The first element is: " + de\_que.getFirst()); * System.***out***.println(""); * } |

### Phương thức getLastElement():

* Chức năng: Lấy ra phần tử ở cuối ArrayDeque.
* Mã nguồn:

|  |
| --- |
| *// get the last element of ArrayDeque*  **public** **static** **void** getLastElement(Deque<Integer> de\_que) {  *// Displaying the first element*  System.***out***.println("The last element is: " + de\_que.getLast());  System.***out***.println("");  } |

### Phương thức isEmpty():

* Chức năng: Kiểm tra ArrayDeque có rỗng hay không
* Mã nguồn:

|  |
| --- |
| *// Verifying if the ArrayDeque is empty or not*  **public** **static** **void** isEmpty(Deque<Integer> de\_que) {  System.***out***.println("Check Deque empty: " + de\_que.isEmpty());  System.***out***.println("");  } |

### Phương thức toArray():

* Chức năng: Chuyển ArrayDeque về dạng mảng thông thường.
* Mã nguồn:

|  |
| --- |
| *// form an array of the same elements as that of the ArrayDeque*  **public** **static** **void** toArray(Deque<Integer> de\_que) {  *// Creating the array and using toArray()*  Object[] arr = de\_que.toArray();  System.***out***.println("The array is:");  **for** (**int** j = 0; j < arr.length; j++)  System.***out***.print(arr[j] + "\t");  System.***out***.println("");  System.***out***.println("");  } |

## Nghiên cứu về ConcurrentLinkedDeque trong Java

Là một phần của Java Collection Framework, thực hiện giao diện Bộ sưu tập và lớp AbstractCollection

Được sử dụng để triển khai Deque với sự giúp đỡ của LinkedList.

Một số phương thức của ConcurrentLinkedDeque:

### Phương thức show():

* Chức năng : Hiển thị lên màn hình console.
* Mã nguồn:

|  |
| --- |
| * *// Displaying the ConcurrentLinkedDeque* * **public** **static** **void** showCld(Deque<Integer> cld) { * *// Displaying the existing LinkedDeque* * System.***out***.println("Initial Elements in" + "the LinkedDeque: " + cld); * System.***out***.println(""); * } |

### Phương thức add():

* Chức năng: thêm 1 hoặc nhiều phần tử ( do người dùng nhập) và hiển thị lên Console.
* Mã nguồn :

|  |
| --- |
| * *// add element into the Deque* * **public** **static** Deque<Integer> addCld(Deque<Integer> cld) { * System.***out***.println("Add value for the ConcurrentLinkedDeque (If value = 0 => Stop)."); * **while** (**true**) { * System.***out***.print("Add value: "); * **int** addValue = *sc*.nextInt(); * **if** (addValue == 0) * **break**; * cld.add(addValue); * } * *showCld*(cld); * **return** cld; * } |

### Phương thức addFirst():

* Chức năng : Dùng để thêm 1 hoặc nhiều phần tử ( do người dùng nhập) được nhập vào đầu của deque.
* Mã nguồn:

|  |
| --- |
| * *// addFirstCld() method to add element to the beginning of the Deque* * **public** **static** Deque<Integer> addFirstCld(Deque<Integer> cld) { * *// n is the number of digits used for use with the addFirst () method* * System.***out***.print("Number of value to use with addFirst: "); * **int** n = *sc*.nextInt(); * *// Adding elements at the begining* * System.***out***.println("Type value you want to use with addFirst(): "); * **for** (**int** i = 0; i < n; i++) { * cld.addFirst(*sc*.nextInt()); * } * **return** cld; * } |

### Phương thức addLast:

* Chức năng : Dùng để thêm 1 hoặc nhiều phần tử ( do người dùng nhập) được nhập vào cuối của deque.
* Mã nguồn :

|  |
| --- |
| * *// addLastCld() method to add element to the end of the Deque* * **public** **static** Deque<Integer> addLastCld(Deque<Integer> cld) { * *// n is the number of digits used for use with the addlast () method* * System.***out***.print("Number of value to use with addLast: "); * **int** n = *sc*.nextInt(); * *// Adding elements at the begining* * System.***out***.println("Type value you want to use with addLast(): "); * **for** (**int** i = 0; i < n; i++) { * cld.addLast(*sc*.nextInt()); * } * **return** cld; * } |

### Phương thức getFirst():

* Chức năng : Lấy, nhưng không loại bỏ, phần tử đầu tiên của deque này hiển thị lên Console.
* Mã Nguồn :

|  |
| --- |
| * **public** **static** **void** getFirstCld(Deque<Integer> cld) { * **try** { * *// Displaying the first element* * System.***out***.println("The first element " + "is: " + cld.getFirst()); * } **catch** (Exception e) { * System.***out***.println(e); * } * } |

### Phương thức getLast():

* Chức năng : Lấy, nhưng không loại bỏ, phần tử cuối cùng của deque và hiển thị lên Console.
* Mã nguồn :

|  |
| --- |
| * **public** **static** **void** getLastCld(Deque<Integer> cld) { * **try** { * *// Displaying the Last element* * System.***out***.println("The Last element " + "is: " + cld.getLast()); * } **catch** (Exception e) { * *// Displaying the Exception* * System.***out***.println(e); * } * } |

### Phương thức offer():

* Chức năng: Chèn 1 phần tử vào đuôi của deque .
* Mã nguồn :

|  |
| --- |
| **public** **static** **void** offerCld(Deque<Integer> cld) {  System.***out***.print("Value to use with offer: ");  **int** x = *sc*.nextInt();  *// Using offer() to add elements*  cld.offer(x);  System.***out***.println("");  *// Displaying the ArrayDeque*  *showCld*(cld);  } |

### Phương thức offerFirst():

* Chức năng: Chèn phần tử được nhập ở phía trước của deque.
* Mã nguồn:

|  |
| --- |
| * **public** **static** **void** offerFirstCld(Deque<Integer> cld) { * System.***out***.print("Value to use with offerFirst: "); * **int** y = *sc*.nextInt(); * *// Displaying the First element* * **try** { * cld.offerFirst(**null**); * } **catch** (Exception e) { * System.***out***.println(e); * } * */\** * *\* Insert an element at the front of the deque* * *\*/* * **if** (cld.offerFirst(y)) { * *// Displaying the First element* * System.***out***.println("The Inserted element is: " + cld.getFirst()); * } * *// Displaying the Deque* * System.***out***.println("Elements in Deque: " + cld); * *// Displaying the First element* * System.***out***.println("The First element is: " + cld.getFirst()); * } |

### Phương thức offerLast():

* Chức năng : Chèn phần tử được nhập ở phía cuối của deque.
* Mã nguồn:

|  |
| --- |
| * **public** **static** **void** offerLastCld(Deque<Integer> cld) { * System.***out***.print("Value to use with offerLast: "); * **int** z = *sc*.nextInt(); * **try** { * cld.offerLast(**null**); * } **catch** (Exception e) { * System.***out***.println(e); * } * */\** * *\* Insert an element at the tail of the deque* * *\*/* * **if** (cld.offerLast(z)) { * *// Displaying the Last element* * System.***out***.println("The Inserted element is: " + cld.getLast()); * } * *// Displaying the Deque* * System.***out***.println("Elements in Deque: " + cld); * *// Displaying the Last element* * System.***out***.println("The Last element is: " + cld.getLast()); * } |

### Phương thức peekFirst():

* Chức năng: Lấy, nhưng không loại bỏ, phần tử đầu tiên của deque này hoặc trả về null nếu deque này trống.
* Mã nguồn :

|  |
| --- |
| * **public** **static** **void** peekFirstCld(Deque<Integer> cld) { * *// Displaying the first element* * System.***out***.println("First Element in" + "the LinkedDeque: " + cld.peekFirst()); * } |

### Phương thức peekLast

* Chức năng: Lấy, nhưng không loại bỏ, phần tử cuối cùng của deque này hoặc trả về null nếu deque này trống.
* Mã nguồn:

|  |
| --- |
| * **public** **static** **void** peekLastCld(Deque<Integer> cld) { * *// Displaying the last element* * System.***out***.println("Last Element in" + "the LinkedDeque: " + cld.peekLast()); * } |

## Nghiên cứu về LinkedBlockingDeque trong Java

Một số phương thức của LinkedBlockingDeque:

### Phương thức show():

* Chức năng: Hiển thị lên màn hình console
* Mã nguồn:

|  |
| --- |
| * *// showLBD() method is used to show LinkedBlockingDeque* * **public** **static** **void** showLBD(LinkedBlockingDeque<Integer> LBD) **throws** IllegalStateException { * System.***out***.println("LinkedBlockingDeque: " + LBD); * } |

### Phương thức add():

* Chức năng: thêm một hoặc nhiều phần tử vào LinkedBlockingDeque do người dùng nhập từ bàn phím
* Mã nguồn:

|  |
| --- |
| * *// addLBD() method is used to inserts the specified element at the end of this* * *// deque* * **public** **static** Deque<Integer> addLBD(LinkedBlockingDeque<Integer> LBD) **throws** IllegalStateException { * *// Use add() method to add elements into the LinkedBlockingDeque* * System.***out***.println("Add value for the LinkedBlockingDeque (If value = 0 => Stop)."); * **while** (**true**) { * System.***out***.print("Add value: "); * **int** addValue = *sc*.nextInt(); * **if** (addValue == 0) * **break**; * LBD.add(addValue); * } * *showLBD*((LinkedBlockingDeque<Integer>) LBD); * **return** LBD; * } |

### Phương thức addFirst():

* Chức năng: thêm a (a do người dùng nhập) phần tử mới lên đầu LinkedBlockingDeque do người dùng nhập từ bàn phím
* Mã nguồn:

|  |
| --- |
| * *// addFirstLBD() method inserts the specified element at the front of this deque* * **public** **static** LinkedBlockingDeque<Integer> addFirstLBD(LinkedBlockingDeque<Integer> LBD) * **throws** IllegalStateException { * *// n is the number of digits used for use with the addfirst () method* * System.***out***.print("Number of the element ​​to insert at the front of deque: "); * **int** a = *sc*.nextInt(); * *// Add elements to front of LinkedBlockingDeque* * System.***out***.println("Type values: "); * **for** (**int** i = 0; i < a; i++) { * System.***out***.print("Add value: "); * LBD.addFirst(*sc*.nextInt()); * } * **return** LBD; * } |

### Phương thức addLast():

* Chức năng: thêm b (b do người dùng nhập) phần tử mới vào cuối ArrayDeque do người dùng nhập từ bàn phím
* Mã nguồn:

|  |
| --- |
| * *// addLastLBD() method inserts the specified element at the end of this deque* * **public** **static** LinkedBlockingDeque<Integer> addLastLBD(LinkedBlockingDeque<Integer> LBD) * **throws** IllegalStateException { * *// n is the number of digits used for use with the addfirst () method* * System.***out***.print("Number of the element ​​to insert at the end of deque: "); * **int** b = *sc*.nextInt(); * *// Add elements to front of LinkedBlockingDeque* * System.***out***.println("Type values: "); * **for** (**int** i = 0; i < b; i++) { * System.***out***.print("Add value: "); * LBD.addLast(*sc*.nextInt()); * } * **return** LBD; * } |

### Phương thức clear():

* Chức năng: xoá hết phần tử của LinkedBlockingDeque (nhưng không xoá LinkedBlockingDeque)
* Mã nguồn:

|  |
| --- |
| * *// clearLBD() method atomically removes all of the elements from this deque* * **public** **static** LinkedBlockingDeque<Integer> clearLBD(LinkedBlockingDeque<Integer> LBD) **throws** IllegalStateException { * LBD.clear(); * *showLBD*(LBD); * **return** LBD; * } |

### Phương thức contains():

* Chức năng: Kiểm tra 1 giá trị có tồn tại trong LinkedBlockingDeque hay không
* Mã nguồn:

|  |
| --- |
| * *// containsLBD() method returns true if this deque contains the specified* * *// element.* * **public** **static** **void** containsLBD(LinkedBlockingDeque<Integer> LBD) **throws** IllegalStateException { * System.***out***.print("\nCheck Contains method of Linked Blocking Deque with value: "); * Scanner sc = **new** Scanner(System.***in***); * **int** c = sc.nextInt(); * **if** (LBD.contains(c)) { * System.***out***.println("Linked Blocking Deque contains " + c); * } **else** { * System.***out***.println("Linked Blocking Deque does not contain " + c); * } * sc.close(); * } |

### Phương thức getElement():

* Chức năng: Lấy ra giá trị đầu của LinkedBlockingDeque
* Mã nguồn:

|  |
| --- |
| * *// getElementLBD() method is used to retrieves, but does not remove, the head of* * *// the queue represented by this deque* * **public** **static** **void** getElementLBD(LinkedBlockingDeque<Integer> LBD) **throws** IllegalStateException { * System.***out***.println("\nLinked Blocking Deque: " + LBD); * *// prints the first element* * System.***out***.println("Linked Blocking Deque front element: " + LBD.element()); * } |

### Phương thức getFirstElement():

* Chức năng: Lấy ra giá trị đầu của LinkedBlockingDeque
* Mã nguồn:

|  |
| --- |
| * *// getFirstElementLBD() method retrieves, but does not remove, the first element* * *// of this deque* * **public** **static** **void** getFirstElementLBD(LinkedBlockingDeque<Integer> LBD) **throws** IllegalStateException { * *// prints the first element* * System.***out***.println("Linked Blocking Deque first element: " + LBD.getFirst()); * } |

### Phương thức getLastElement():

* Chức năng: Lấy ra giá trị cuối của LinkedBlockingDeque
* Mã nguồn:

|  |
| --- |
| * *// getLastElementLBD() method retrieves, but does not remove, the last element* * *// of this deque* * **public** **static** **void** getLastElementLBD(LinkedBlockingDeque<Integer> LBD) **throws** IllegalStateException { * *// prints the last element* * System.***out***.println("Linked Blocking Deque last element: " + LBD.getLast()); * } |

### Phương thức iterator():

* Chức năng: Trả về 1 một vòng lặp các phần tử của LinkedBlockingDeque
* Mã nguồn:

|  |
| --- |
| * *// iteratorLBD() method returns an iterator over the elements in this deque in* * *// proper sequence* * **public** **static** **void** iteratorLBD(LinkedBlockingDeque<Integer> LBD) { * *// Call iterator() method of LinkedBlockingDeque* * Iterator<Integer> iteratorVals = LBD.iterator(); * *// Print elements of iterator* * *// created from PriorityBlockingQueue* * System.***out***.println("\nThe iterator values" + " of LinkedBlockingDeque are:"); * *// prints the elements using an iterator* * **while** (iteratorVals.hasNext()) { * System.***out***.print(iteratorVals.next() + "\t"); * } * System.***out***.println(""); * } |

### Phương thức peek():

* Chức năng: Lấy ra phần tử ở đầu LinkedBlockingDeque
* Mã nguồn:

|  |
| --- |
| * *// This method retrieves, but does not remove, the head of the queue represented* * **public** **static** **void** peekLBD(LinkedBlockingDeque<Integer> LBD) { * *// prints the Deque after removal* * System.***out***.println("Front element in Deque: " + LBD.peek()); * } |

### Phương thức remove():

* Chức năng: Xoá phần tử đầu LinkedBlockingDeque
* Mã nguồn:

|  |
| --- |
| * *// remove() method retrieves and removes the head of the queue represented by* * *// this deque* * **public** **static** **void** removeLBD(LinkedBlockingDeque<Integer> LBD) **throws** InterruptedException { * *// removes the front element* * LBD.remove(); * System.***out***.println("Linked Blocking Deque after using remove() method: " + LBD); * } |

### Phương thức size():

* Chức năng: Đếm số phần tử trong LinkedBlockingDeque
* Mã nguồn:

|  |
| --- |
| * *// size() method returns the number of elements in this deque* * **public** **static** **void** sizeLBD(LinkedBlockingDeque<Integer> LBD) **throws** InterruptedException { * *// prints the Deque after removal* * System.***out***.println("Size of Linked Blocking Deque: " + LBD.size()); * System.***out***.println(""); * } |

### Phương thức take():

* Chức năng: Phương thức này loại bỏ phần tử ở đầu hàng đợi
* Mã nguồn:

|  |
| --- |
| * *// This method retrieves and removes the head of the queue represented by this* * *// deque* * **public** **static** **void** takeLBD(LinkedBlockingDeque<Integer> LBD) **throws** InterruptedException { * *// print Dequeue* * System.***out***.println("\nLinked Blocking Deque: " + LBD); * *// removes the front element and prints it* * System.***out***.println("Head of Linked Blocking Deque: " + LBD.take()); * *// prints the Deque* * System.***out***.println("Linked Blocking Deque after using take() method: " + LBD); * } |

### Phương thức toString():

* Chức năng: Phương thức này trả về kiểu dữ liệu String cho hàng đợi (Nếu có)
* Mã nguồn:

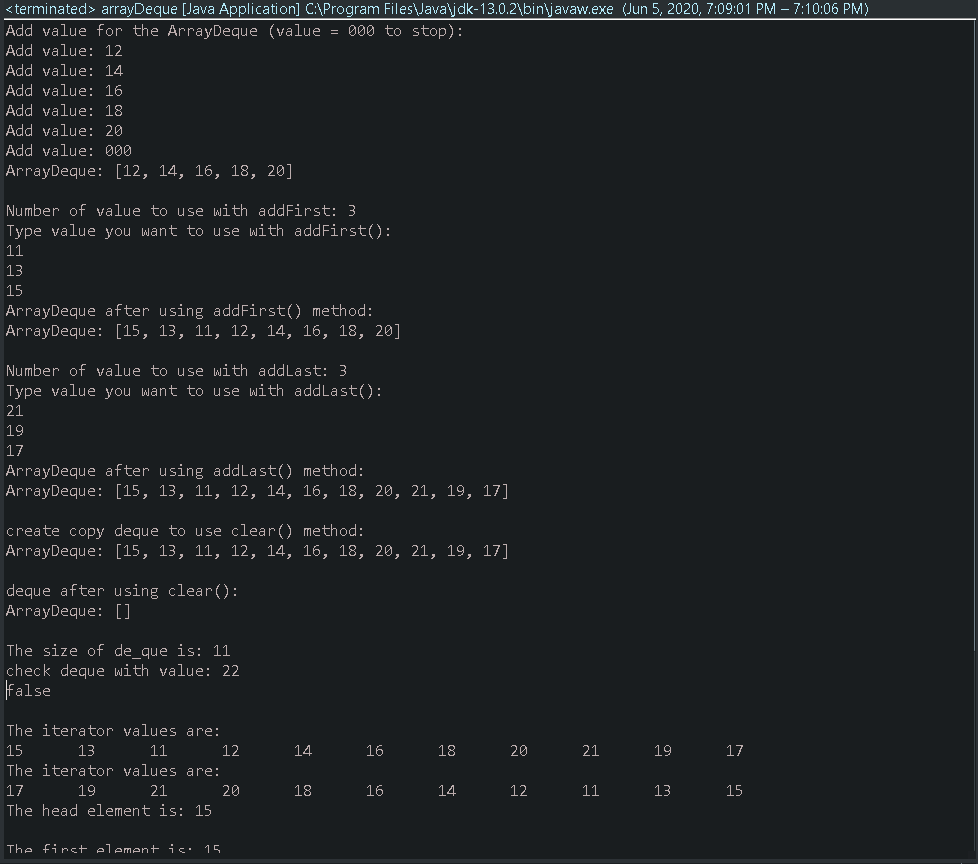
|  |
| --- |
| * *// This method returns a string representation of this collection* * **public** **static** **void** toStringLBD(LinkedBlockingDeque<Integer> LBD) { * System.***out***.println("LBD in string " + LBD.toString()); * } |

# Chạy Demo Các Phương Thức API

## Chạy chương trình

### Kết quả chạy các method của ArrayDeque

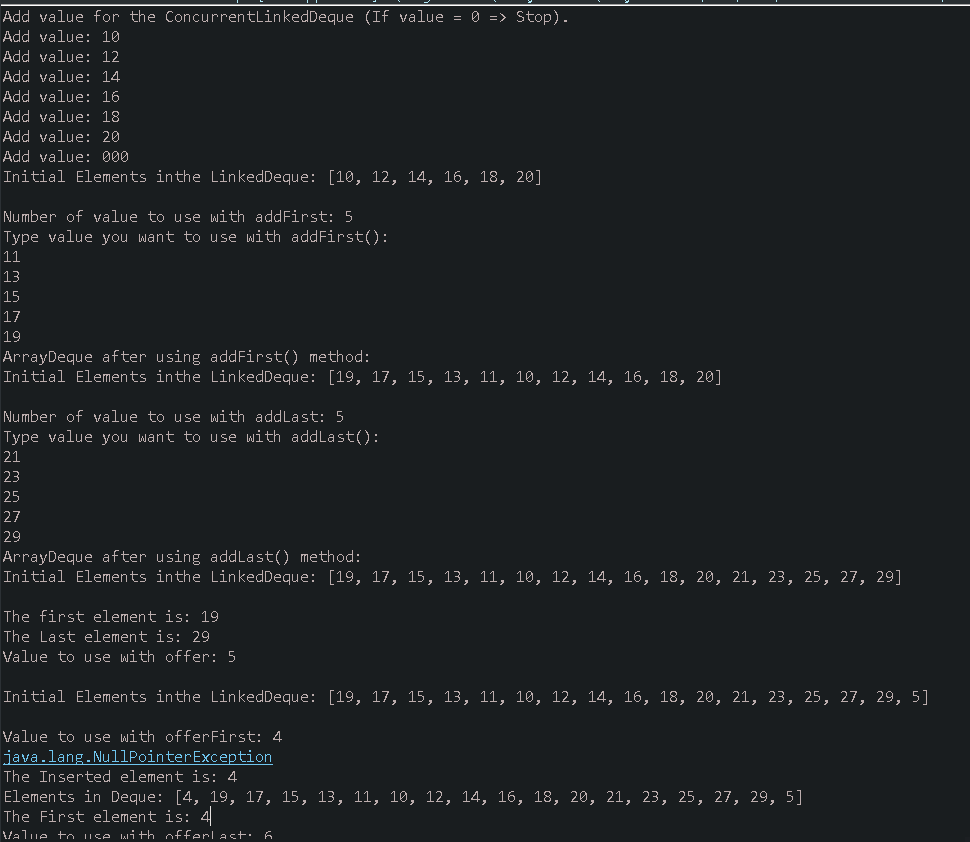
* Mô tả:
  + Khởi tạo ArrayDeque bằng phương thức intializingIntegerDeque()
  + Nhập vào hang đợi bằng phương thức addDeque() các giá trị: [12, 14, 16, 18, 20]
  + Sử dụng phương thức addFirst() thêm 3 giá trị 11, 13, 15
  + Sử dụng phương thức addLast() thêm 3 giá trị 21, 19 , 17
  + Tạo ra một bản sao ArrayDeque rồi sử dụng hàm clear() để xoá các phần tử trong ArrayDeque(bản sao)
  + Sử dụng hàm sizeDeque() để tính số phần tử có trong ArrayDeque
  + Sử dụng hàm contains() để kiểm tra giá trị có tồn tại trong ArrayDeque hay không?
  + Sử dụng hàm iteratorDeque() để hiển thị 1 vòng lặp các phần tử của ArrayDeque
  + Sử dụng hàm descendingIteratorDeque() để hiển thị 1 vòng lặp ngược (từ cuối lên) các phần tử của ArrayDeque
  + Sử dụng hàm fetchHeadArrayDeque() để tìm phần tử đầu của ArrayDeque
  + Sử dụng hàm getFirstElement() để lấy ra phần tử đầu tiên của ArrayDeque
  + Sử dụng hàm getLastElement() để lấy ra phần tử cuối cùng của ArrayDeque
  + Tạo ra một ArrayDeque mới rồi sử dụng hàm isEmpty() để kiểm tra ArrayDeque có rỗng hay không
  + Sử dụng hàm toArray() để chuyển đổi và in ArrayDeque về dạng mảng thông thường

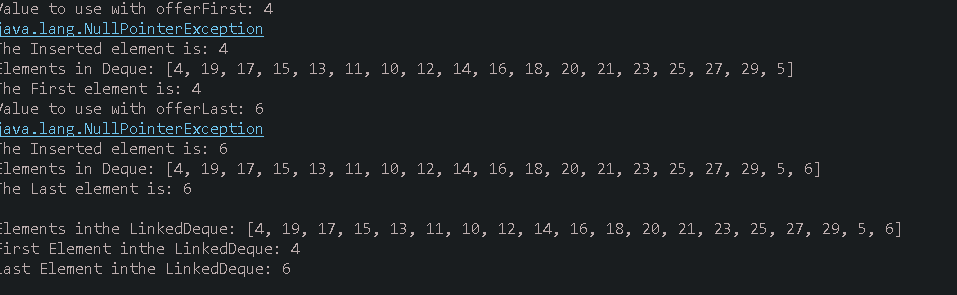




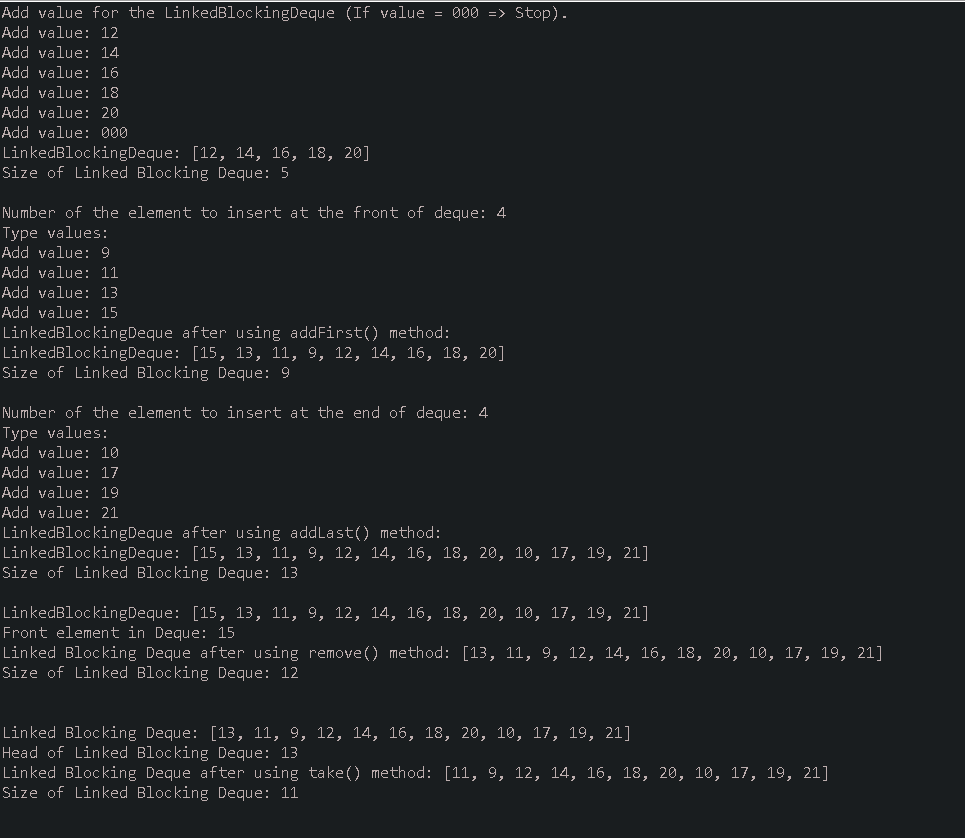
### Kết quả chạy các method của ConcurrentLinkedDeque

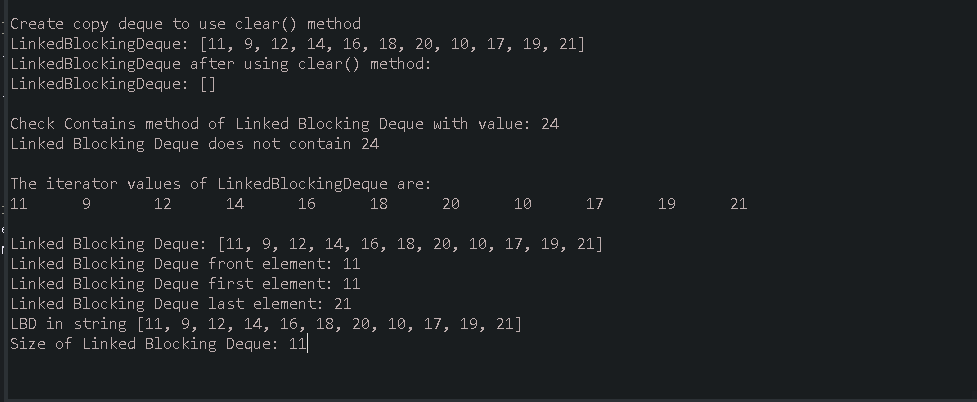
* Mô tả:
  + Khởi tạo ConcurrentLinkedDeque bằng phương thức intializingIntegerCld()
  + Nhập vào hang đợi bằng phương thức addCld () các giá trị: [12, 14, 16, 18, 20]
  + Sử dụng phương thức addFirstCld() thêm 3 giá trị 11, 13, 15
  + Sử dụng phương thức addLastCld() thêm 3 giá trị 21, 19 , 17
  + Sử dụng hàm getFirstCld() để lấy ra phần tử đầu tiên của ConcurrentLinkedDeque
  + Sử dụng hàm getLastCld() để lấy ra phần tử cuối cùng của ConcurrentLinkedDeque
  + Sử dụng hàm offerCld() để chèn phần tử vào phần đuôi của ConcurrentLinkedDeque
  + Sử dụng hàm offerFirstCld() để chèn phần tử vào phần trước của ConcurrentLinkedDeque
  + Sử dụng hàm offerLastCld() để chèn phần tử vào phần cuối cùng của ConcurrentLinkedDeque
  + Sử dụng hàm peekFirst() để trả về giá trị đầu tiên của ConcurrentLinkedDeque(không xoá), trả về null nếu ConcurrentLinkedDeque rỗng
  + Sử dụng hàm peekLast() để trả về giá trị cuối cùng của ConcurrentLinkedDeque(không xoá), trả về null nếu ConcurrentLinkedDeque rỗng





### Kết quả chạy các method của LinkedBlockingDeque





**Tài liệu tham khảo**

(Các tài liệu tham khảo để hoàn thành bài tập lớn)

<https://www.geeksforgeeks.org/arraydeque-in-java/>

<https://www.geeksforgeeks.org/concurrentlinkeddeque-in-java-with-examples/>

<https://www.geeksforgeeks.org/linkedblockingdeque-in-java-with-examples/>

(Slide bài giảng giảng viên - Ths Hoàng Quang Huy biên soạn)

**--------------------------------HẾT --------------------------------**