УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Кафедра інформаційних технологій та програмної інженерії

КУРСОВА РОБОТА

з Об’єктно-орієнтованого програмування

на тему: Клас для обслуговування структур типу «Дек»

Студента II курсу _____ групи

галузі знань 12 «Інженерія програмного забезпечення»

спеціальності 121 «Інженерія програмного забезпечення»

Попович.П.П.

(прізвище та ініціали)

Керівник _____к.т.н., Пашкевич О.П.

Національна шкала _____

Кількість балів _____ Оцінка: ECTS _____

м. Івано-Франківськ

2021

Університет Короля Данила
назва вищого навчального закладу

Кафедра інформаційних технологій та програмної інженерії

Дисципліна об'єктно орієнтоване програмування

Спеціальність 121 «Інженерія програмного забезпечення»

Курс II Група ІПЗс-2019 Семестр 4

ЗАВДАННЯ

на курсовий проект (роботу) студента

Попович Петро Петрович
Прізвище, ім’я, по-батькові

1. Тема проекту (роботи) Клас для обслуговування структур типу «Дек»

2. Строк здачі студентом закінченого проекту (роботи)

3. Вихідні дані до проекту (роботи)

4. Зміст роботи (перелік питань по розділах, які потрібно розробити) _____

_____

_____

5.Дата видачі завдання

_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсового проекту (роботи)	Строк виконання	Примітки
	Вибір теми власного варіанту курсової роботив в методичці. Опрацювання відповідної літератури	22.04.2021	
1	Узгодження постановки задачі	29.04.2021	
	Пошук та вивчення джерел з питань курсової роботи	05.05.2021	
	Розробка сценарію роботи програми	09.05.2021	
	Розробка алгоритму рішення задачі	11.05.2021	
	Розробка інтерфейсу користувача	14.05.2021	
	Розробка інформаційного забезпечення	14.05.2021	
	Розробка програмного забезпечення	15.05.2021	
	Тестування програми	18.05.2021	
	Аналіз результатів.Оформлення.	21.05.2021	
	Захист	24.05.2021	

Студент _____ _____
Підпис Прізвище, ініціали

Керівник _____ _____
Підпис Прізвище, ініціали

«____» _____ 202___ р.

ЗМІСТ

Вступ

Deque - це лінія з подвійними кінцями. Це може додавати і видаляти елементи даних з голови або хвоста структури даних. Його можна використовувати як FIFO або як LIFO. Уявлення FIFO і LIFO - це черга і стек в Java відповідно.

Так відчувається схематичність роботи. Рухаючись далі, ми включили в двосторонню чергу кілька методів. Подивимося на це**.**

Черги представляють структуру даних, що працює за принципом FIFO (first in - first out). Тобто чим раніше був доданий елемент в колекцію, тим раніше він з неї видаляється. Це стандартна модель односпрямованої черзі. Однак бувають і двонаправлені - тобто такі, в яких ми можемо додати елемент не тільки в початку, а й в кінець. І відповідно видалити елемент не тільки з кінця, але і з початку.

Особливістю класів черг є те, що вони реалізують спеціальні інтерфейси Queue або Deque.

Інтерфейс Queue

Узагальнений інтерфейс Queue <E> розширює базовий інтерфейс Collection і визначає поведінку класу в якості односпрямованої черзі. Свою функціональність він розкриває через такі методи: Таким чином, наявність методів pop і push дозволяє класам, які реалізують цей елемент, діяти в якості стека. У той же час наявний функціонал також дозволяє створювати двонаправлені черзі, що робить класи, які застосовують даний інтерфейс, досить гнучкими.

В інформатиці двостороння черга (деквіта, часто скорочується до deque, яскраво виражена колода) - це абстрактний тип даних, який узагальнює чергу, для якого елементи можуть бути додані або видалені з передньої (голови) або задньої (хвоста).

Інтерфейси Deque можуть бути реалізовані за допомогою різних типів колекцій, таких як класи LinkedList або ArrayDeque.

Функції, що викликаються у програмі

- equals - порівняння;
- union - об’єднання множин;
- section - переріз множин;
- subtract - віднімання;
- size - кількість елементів;
- empty - повертає значення true, якщо множина пуста;
- swap - обмін значеннями з іншою множиною;
- include - додати новий елемент у множину;
- exclude - вилучити наданий елемент із множини;
- compl - отримати значення, яке є доповненням до наданої множини;
- find - перевірити, чи є наданий елемент у множині;
- accumulate - накопичення суми або добутку для всіх елементів множини;
- for_each - змінювання елементів згідно з наданою процедурою.

Код програми

```
class Deque

{

static final int MAX = 100;

int  arr[];

int  front;

int  rear;

int  size;
```

```
public Deque(int size)

{

arr = new int[MAX];

front = -1;

rear = 0;

this.size = size;

}


/*// Операції на Deque:

void  insertfront(int key);

void  insertrear(int key);

void  deletefront();

void  deleterear();

bool  isFull();

bool  isEmpty();

int  getFront();

int  getRear();*/


// Перевіряє, заповнений Deque чи ні.

boolean isFull()

{

return ((front == 0 && rear == size-1)||
```

```
front == rear+1);
}

// Перевіряє, чи порожній Deque.
boolean isEmpty ()
{
return (front == -1);
}

// Вставляє елемент спереду
void insertfront(int key)
{
// Перевіряє, чи черга повна чи ні
if (isFull())
{
System.out.println("Overflow");
return;
}

// Якщо черга спочатку порожня
if (front == -1)
{
```

```
front = 0;
rear = 0;
}

// front - це перша позиція черги
else if (front == 0)
front = size - 1 ;

else // зменшити передній кінець на "1"
front = front-1;

// вставити поточний елемент у Deque
arr[front] = key ;
}

// функція для вставки елемента на задній частині
// перерахунку.
void insertrear(int key)
{
if (isFull())
{
System.out.println(" Overflow ");
```

```
return;
}

// Якщо черга спочатку порожня
if (front == -1)
{
front = 0;
rear = 0;
}

// rear - це остання позиція черги
else if (rear == size-1)
rear = 0;

// збільшити задній кінець на «1»
else
rear = rear+1;

// Вставити поточний елемент у Deque
arr[rear] = key ;
}
```

```
// Видаляє елемент у передньому кінці Deque
void deletefront()
{
// перевірити, чи порожній Deque чи ні
if (isEmpty())
{
System.out.println("Queue Underflow\n");
return ;
}

// Deque має лише один елемент
if (front == rear)
{
front = -1;
rear = -1;
}
else
// повернутися у початкове положення
if (front == size -1)
front = 0;

else // збільшити front на '1', щоб видалити струм
```

```
// переднє значення від Deque
front = front+1;
}

// Видалити елемент на задньому кінці Deque
void deleterear()
{
if (isEmpty())
{
System.out.println(" Underflow");
return ;
}

// Deque має лише один елемент
if (front == rear)
{
front = -1;
rear = -1;
}
else if (rear == 0)
rear = size-1;
else
```

```
rear = rear-1;
}

// Повертає передній елемент Deque
int getFront()
{
// перевіряємо, чи порожній Deque чи ні
if (isEmpty())
{
System.out.println(" Underflow");
return -1 ;
}
return arr[front];
}

// функція повернення заднього елемента Deque
int getRear()
{
// перевіряємо, чи порожній Deque чи ні
if(isEmpty() || rear < 0)
{
System.out.println(" Underflow\n");
```

```
return -1 ;
}
return arr[rear];
}

// Драйвер-програма для тестування вищезгаданої функції
public static void main(String[] args)
{

Deque dq = new Deque(5);

System.out.println("вставний елемент на задньому кінці  : 5 ");
dq.insertrear(5);

System.out.println("вставний елемент на задньому кінці : 10 ");
dq.insertrear(10);

System.out.println("отримати задній елемент : "+ dq.getRear());

dq.deleterear();
System.out.println("Після видалення заднього елемента стає новий задній елемент : " +
```

```
dq.getRear());

System.out.println("вставний елемент на передньому кінці");
dq.insertfront(15);

System.out.println("отримати передній елемент: " +dq.getFront());

dq.deletefront();

System.out.println("Після видалення переднього елемента стають нові : " +
+  dq.getFront());

}
}
```

Deque або Double Ended Quee - це узагальнена версія структури даних Queue, яка дозволяє вставляти та видаляти з обох кінців. У попередній публікації ми обговорювали введення deque. Тепер у цій публікації ми бачимо, як ми реалізуємо deque за допомогою кругового масиву.

Операціїз Deque:

В основному наступні чотири основні операції виконуються в черзі:

insetFront () : Додає елемент в передній частині Deque.

insertRear () : Додає елемент у задній частині Deque.

deleteFront () : Видаляє елемент з передньої частини Deque.

deleteRear () : видаляє елемент із задньої сторони Deque.

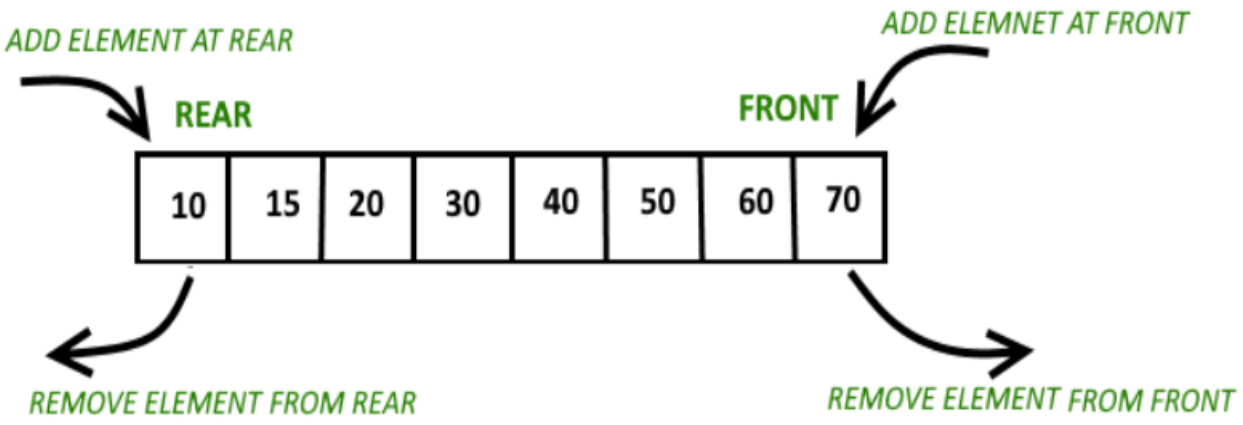
На додаток до вищезазначених операцій, також підтримуються наступні операції

getFront (): Отримує передній елемент із черги.

getRear () : Отримує останній елемент із черги.

isEmpty () : перевіряє, чи порожній **Deque** .

isFull () : перевіряє, заповнено **чисельність Deque** .



Круговий масив реалізації deque

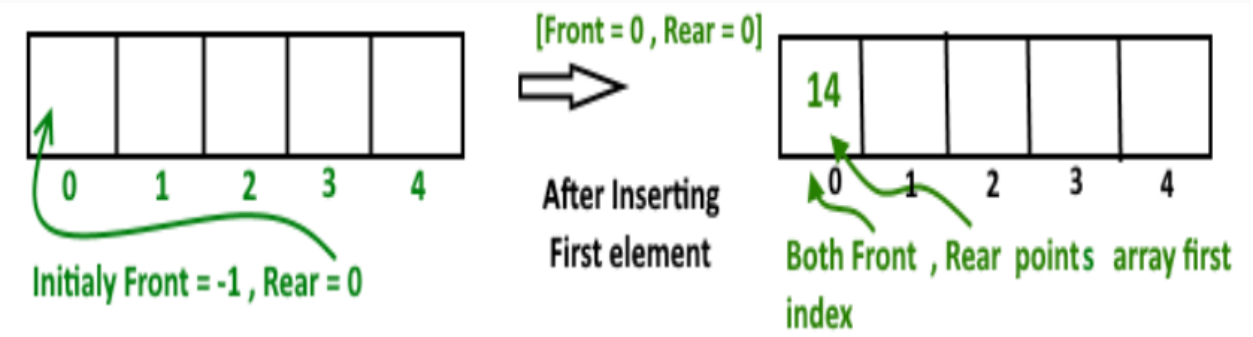
Для реалізації deque нам потрібно відстежувати два індекси, передній і задній. Ми ставимо в чергу (штовхаємо) елемент із заднього або переднього кінця черги і виводимо (висуваємо) предмет як із заднього, так і з переднього кінця.

Робота

1. Створіть порожній масив 'arr' розміром 'n'

ініціалізуйте **front = -1** , **rear = 0**

Вставка першого елемента в deque, спереду або ззаду призведе до того самого результату.



Після вставки **Передні** точки = 0 і **Задні** точки = 0

Вставлено елементи в задній кінець

а). Спочатку ми перевіряємо deque, повна чи ні

б). ЯКЩО задній == size -1

потім повторно ініціалізуйте Rear = 0;

Інший приріст ззаду на '1'

і натисніть поточну клавішу в Arr [rear] = key

Спереду залишаються незмінними.

Вставте елементи в передній кінець

а). Спочатку ми перевіряємо deque, якщо він повний чи ні

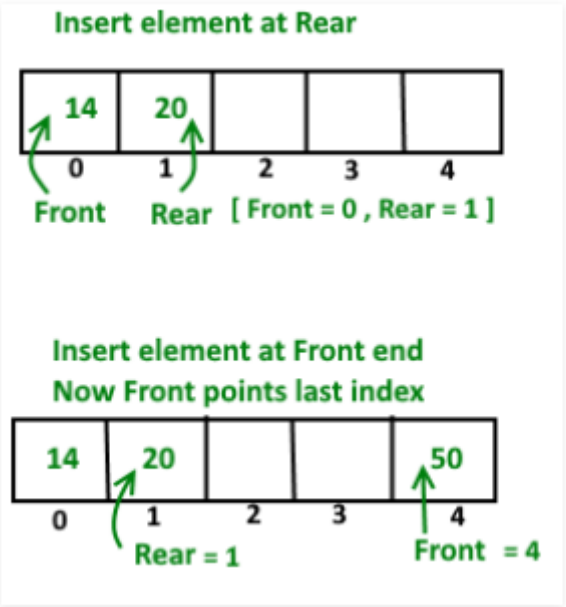
b). ЯКЩО спереду == 0 || початкове положення, перемістіть Front

до точок останнього індексу масиву

front = size - 1 В

іншому випадку зменшено фронт на '1' і натисніть

поточну клавішу в Arr [Front] = клавіша Rear залишається незмінною.



Видалити елемент із заднього кінця

а). спочатку перевірте значення пустого чи ні

б). Якщо deque має лише один елемент

спереду = -1; тил = -1;

Інакше IF Rear вказує на перший індекс масиву

це означає, що ми повинні рухатися назад до точок

останній індекс [тепер перший вставлений елемент в

передній кінець стає тильним кінцем]

rear= size-1;

Інакше || зменшити rear на '1'

rear= rear-1;

Видалити елемент з передньої частини

а). спочатку перевірте значення пустого чи ні

б). Якщо deque має лише один елемент

спереду = -1; `rear` = -1;

Else IF front вказує на останній індекс масиву

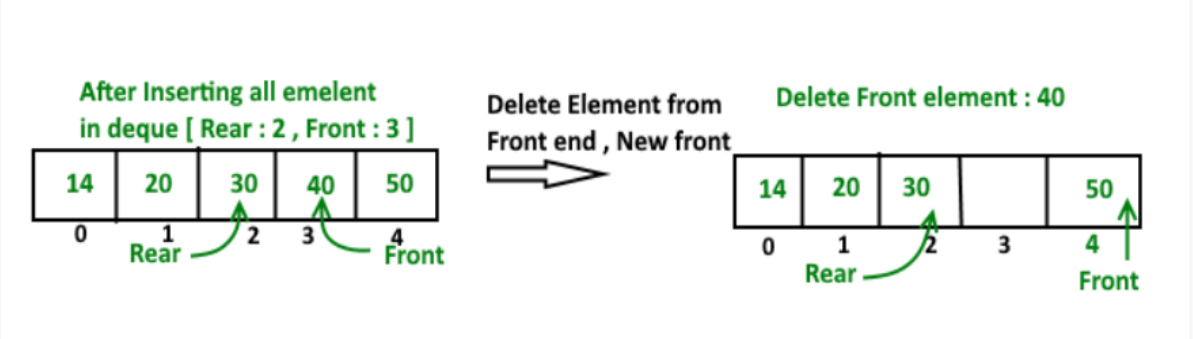
це означає, що у нас більше немає елементів у масиві, тому

рухаємося спереду до пунктів першого індексу масиву

спереду = 0;

Інакше || збільшити спереду на '1'

front = front + 1;



Функції, що викликаються у програмі

E element (): повертає, але не видаляє, елемент з початку черги. Якщо чергу порожня, генерує виняток NoSuchElementException

boolean offer (E obj): додає елемент obj в кінець черги. Якщо елемент вдало доданий, повертає true, інакше - false

E peek (): повертає без видалення елемент з початку черги. Якщо чергу порожня, повертає значення null

E poll (): повертає з видаленням елемент з початку черги. Якщо чергу порожня, повертає значення null

E remove (): повертає з видаленням елемент з початку черги. Якщо чергу порожня, генерує виняток NoSuchElementException

Таким чином, у всіх класів, які реалізують даний інтерфейс, буде метод offer для додавання в чергу, метод poll для вилучення елемента з голови черги, і методи peek і element, що дозволяють просто отримати елемент з голови черги.

Інтерфейс Deque

Інтерфейс Deque розширює вищеописаний інтерфейс Queue і визначає поведінку двобічної черзі, яка працює як звичайна однонаправлена черга, або як стек, що діє за принципом LIFO (останній увійшов - перший вийшов).

Інтерфейс Deque визначає наступні методи:

void addFirst (E obj): додає елемент в початок черги

void addLast (E obj): додає елемент obj в кінець черги

E getFirst (): повертає без видалення елемент з голови черги. Якщо черга порожня, генерує виняток NoSuchElementException

E getLast (): повертає без видалення останній елемент черги. Якщо черга порожня, генерує виняток NoSuchElementException

boolean offerFirst (E obj): додає елемент obj в самий початок черги. Якщо елемент вдало доданий, повертає true, інакше - false

boolean offerLast (E obj): додає елемент obj в кінець черги. Якщо елемент вдало доданий, повертає true, інакше - false

E peekFirst (): повертає без видалення елемент з початку черги. Якщо черга порожня, повертає значення null

E peekLast (): повертає без видалення останній елемент черги. Якщо черга порожня, повертає значення null

E pollFirst (): повертає з видаленням елемент з початку черги. Якщо черга порожня, повертає значення null

E pollLast (): повертає з видаленням останній елемент черги. Якщо черга порожня, повертає значення null

E pop (): повертає з видаленням елемент з початку черги.Якщо черга порожня, генерує виняток NoSuchElementException

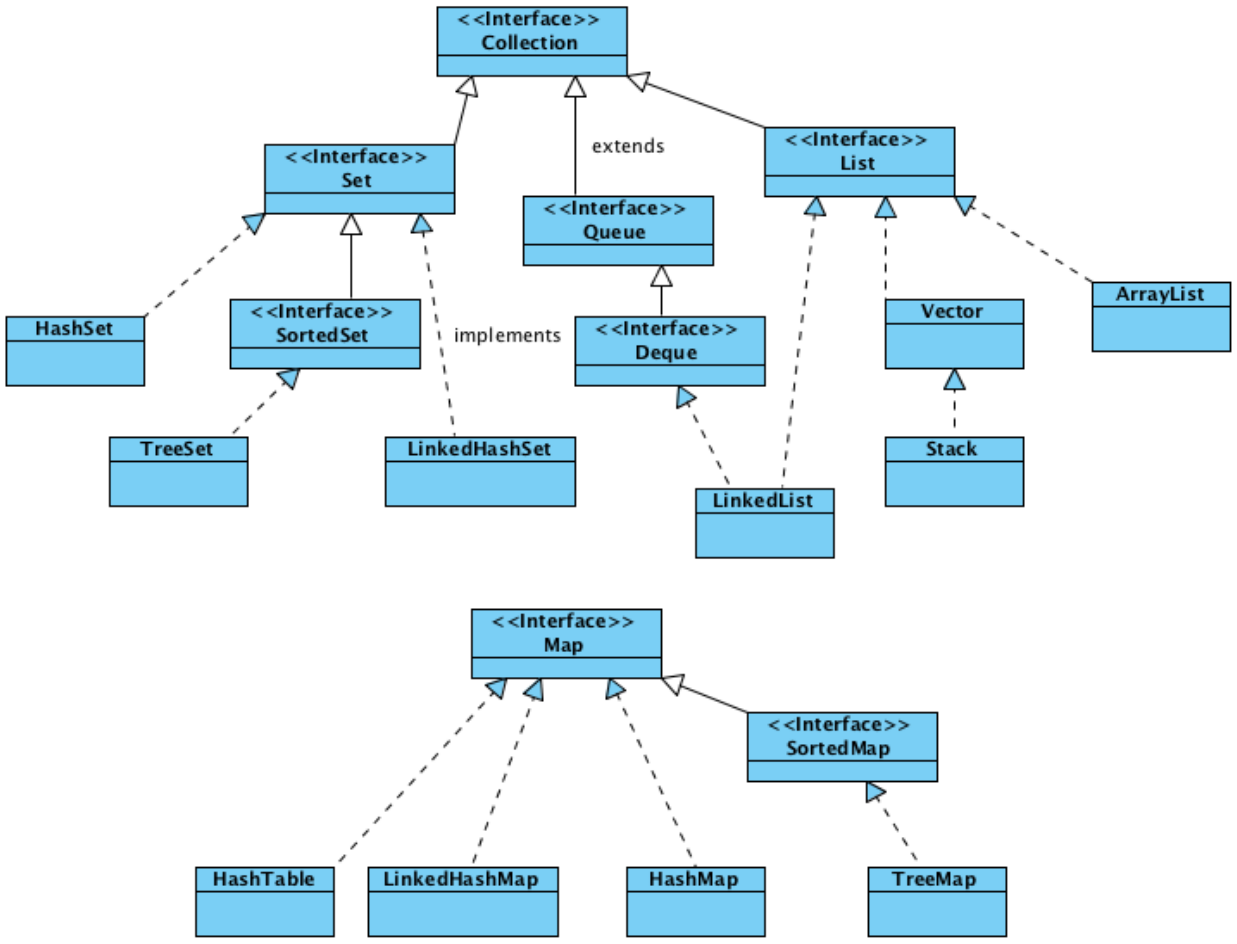
void push (E element): додає елемент в самий початок черги

E removeFirst (): повертає з видаленням елемент з початку черга. Якщо черга порожня, генерує виняток NoSuchElementException

E removeLast (): повертає з видаленням елемент з кінця черги. Якщо черга порожня, генерує виняток NoSuchElementException

boolean removeFirstOccurrence (Object obj): видаляє перший зустрінутий елемент obj з черги. Якщо видалення проізшло, то повертає true, інакше повертає false.

boolean removeLastOccurrence (Object obj): видаляє останній зустрінутий елемент obj з черги. Якщо видалення проізшло, то повертає true, інакше повертає false.



Реалізація масиву змінюваного розміру Deque інтерфейс. У двосторонніх черг масиву немає ніяких обмежень ємності; вони ростуть у міру необхідності, щоб підтримувати використання. Вони не орієнтовані на багатопотоковості виконання; за відсутності зовнішньої синхронізації вони не підтримують паралельний доступ багаторазовими потоками. Забороняються нульові елементи. Цей клас, ймовірно, буде швидше ніж Stack коли використовується в якості стека, і швидше ніж LinkedList коли використовується в якості черзі.

У амортизується постійне час працює більшість операцій ArrayDeque. Винятки включають remove, removeFirstOccurrence, removeLastOccurrence,

contains, iterator.remove (), і об'ємні операції, все з який виконаний в лінійний час.

iterators, повернуті методом iterator цього класу, є збоєм швидко: Якщо двостороння чергу змінюється коли-небудь після того, як iterator створюється, завжди крім через власний метод remove iterator, iterator зазвичай кине a ConcurrentModificationException. Таким чином, перед обличчям паралельної модифікації, iterator перестав працювати швидко і чисто, замість того, щоб ризикнути довільним, недетермінованим поведінкою в невизначений час у майбутньому.

Відзначте, що поведінка збою швидко iterator не може бути гарантовано, як, взагалі кажучи, неможливо зробити будь-які важкі гарантії в присутності несінхронізіруемой паралельної модифікації. Перестаньте працювати швидко iterators кидають ConcurrentModificationException на основі максимальних зусиль. Тому, було б неправильно записати програму, яка залежала від цього винятку для його правильності: поведінка збою швидко iterators має використовуватися тільки, щоб виявити помилки.

Цей клас і його iterator реалізують всі додаткові методи Collection і Iterator інтерфейси.

Клас java.util.ArrayDeque надає resizable-array і реалізує інтерфейс Deque. Нижче наведено важливі моменти, що стосуються Array Deques.

У запитів масиву немає обмежень по ємності, тому вони ростуть у міру необхідності для підтримки використання.

Вони не є потокобезпечна; при відсутності зовнішньої синхронізації.

Вони не підтримують одночасний доступ декількома потоками.

Нульові елементи заборонені в масиві deques.

Вони швидше, ніж Stack і LinkedList.

У запитів масиву немає обмежень по ємності, тому вони ростуть у міру необхідності для підтримки використання.

Вони не є потокобезпечна; при відсутності зовнішньої синхронізації.

Вони не підтримують одночасний доступ декількома потоками.

Нульові елементи заборонені в масиві deques.

Вони швидше, ніж Stack і LinkedList.

Цей клас і його итератор реалізують всі необов'язкові методи інтерфейсів Collection і Iterator.

ArrayDeque ()

Цей конструктор використовується для створення порожнього масиву deque з початковою ємністю, достатньої для зберігання 16 елементів.

ArrayDeque (Колекція <? Extends E> c)

Цей конструктор використовується для створення deque, що містить елементи зазначеної колекції.

ArrayDeque (int numElements)

Цей конструктор використовується для створення порожнього масиву deque з початковою ємністю, достатньої для зберігання зазначеної кількості елементів.

Цей метод вставляє вказаний елемент в кінець цієї черги.

Цей метод вставляє вказаний елемент в початок цієї черги.

Цей метод вставляє вказаний елемент в кінець цієї черги.

Цей метод видаляє всі елементи з цієї черги.

Цей метод повертає копію цієї черги.

Цей метод повертає істину, якщо ця дека містить зазначений елемент.

Цей метод повертає ітератор для елементів в цій деці в зворотному послідовному порядку.

Цей метод витягує, але не видаляє заголовок черги, представленої цією декою.

Цей метод витягує, але не видаляє перший елемент цієї черги.

Цей метод витягує, але не видаляє останній елемент цієї черги.

Цей метод повертає true, якщо в цій деці немає елементів.

Цей метод повертає ітератор для елементів в цій deque.

Цей метод вставляє вказаний елемент в кінець цієї черги.

Цей метод вставляє вказаний елемент в початок цієї черги.

Цей метод вставляє вказаний елемент в кінець цієї черги.

Цей метод витягує, але не видаляє заголовок черги, представленої цим deque, або повертає null, якщо цей deque порожній.

Цей метод витягує, але не видаляє перший елемент цієї deque, або повертає null, якщо ця deque порожня.

Цей метод витягує, але не видаляє останній елемент цієї черги, або повертає нуль, якщо ця черга порожня.

Цей метод витягує і видаляє заголовок черги, представленої цим deque, або повертає null, якщо цей deque порожній.

Цей метод витягує і видаляє перший елемент цієї черги або повертає нуль, якщо ця черга порожня.

Цей метод витягує і видаляє останній елемент цієї черги або повертає нуль, якщо ця черга порожня.

Цей метод витягує елемент з стека, представленого цим deque.

Цей метод поміщає елемент в стек, представлений цією декою.

Цей метод витягує і видаляє заголовок черги, представленої цим deque.

Цей метод видаляє один примірник зазначеного елемента з цієї черги.

Цей метод витягує і видаляє перший елемент цієї черги.

Цей метод видаляє перше входження зазначеного елемента в цій черзі.

Цей метод витягує і видаляє останній елемент цієї черги.

Цей метод видаляє останнє входження зазначеного елемента в цій черзі.

Цей метод повертає кількість елементів в цій деці.

Цей метод повертає масив, що містить всі елементи в цій deque в правильній послідовності.

Висновок

Deque - це двостороння черга, яка дозволяє нам додавати / видаляти елементи з обох кінців, тобто спереду та ззаду, черги. Deque може бути реалізований за допомогою масивів або пов'язаних списків. Однак у нас також є клас стандартної бібліотеки шаблонів (STL), який реалізує різні операції Deque.

У Java ми маємо інтерфейс Deque, який успадковується від інтерфейсу черги для реалізації Deque. Окрім основних стандартних операцій Deque, цей інтерфейс підтримує різні інші операції, які можна виконувати на Deque.

Зазвичай Deque використовується для додатків, які вимагають додавання / видалення елементів з обох кінців. Він також в основному використовується при плануванні процесорів у багатопроцесорних системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

https://coderlessons.com/tutorials/java-tekhnologii/izuchite-paket-java-util/klass-java-util-arraydeque

https://www.google.com/search?q=%D0%B1%D0%BB%D0%BE%D0%BA+%D1%81%D1%85%D0%B5%D0%BC%D0%B0+deque+java&rlz=1C1SQJL_en&sxsrf=ALeKk029zgMAy5gurtNFA1HA2AR8h7_zgw:1621800138339&tbm=isch&source=iu&ictx=1&fir=2U45yix3cFS9pM%252CJl_9furRxfP1AM%252C_&vet=1&usg=AI4_-kSTFVNPIHrlpGPc67MgpC-8uKVlHg&sa=X&ved=2ahUKEwjAp5GpzODwAhUv-yoKHfHSByMQ9QF6BAgOEAE#imgrc=eB_yExtIlwUHQM

https://www.geeksforgeeks.org/deque-interface-java-example/

https://www.geeksforgeeks.org/implementation-deque-using-circular-array/