

chapter 6

Stacks & Queues

5.1 Stacks :- collection of objects that are inserted and removed according to the Last-in first-out (LIFO)

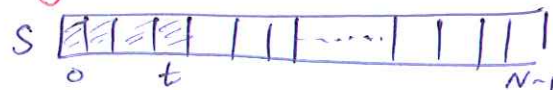
← آخر obj يضاف الى stack و أول obj يخلع منه

* The stack Abstract Data Type (ADT)

- ① push(e) :- يضاف العنصر e الى stack و top
- ② pop() :- يخلع العنصر من stack و top
- ③ top() :- يترجع العنصر في الـ top
- ④ size() :- يترجع عدد العناصر في stack
- ⑤ isEmpty() :- يترجع true اذا كانت فارغة و false اذا لم تكن

operation	output	stack
push(5)	—	(5)
push(3)	—	(5, 3)
pop()	3	(5)
push(7)	—	(5, 7)
pop()	7	(5)
top()	5	(5)
pop()	5	()
isEmpty()	true	()
size()	0	()

* A Simple Array-Based stack Implementation



← يتم تمثيل stack في array S من خلال المتغير integer t

← t هي القيمة التي تحتلها stack في الـ array

← t+1 هي القيمة التي تحتلها stack في الـ array

بناءً على قاعده $t \leftarrow t-1$ على ADT methods implementation

Algorithm size(): return $t+1$ $O(1)$	Algorithm isEmpty(): return $(t < 0)$ $O(1)$	Algorithm top(): if isEmpty() then throw an EmptyStackException return $S[t]$ $O(1)$
--	--	--

Algorithm push(e): $O(1)$

if size() = N then

throw a FullStackException

$t \leftarrow t+1$ $\rightarrow$ t ففوة

$S[t] \leftarrow e$ $\rightarrow$ ضيف e في مكان t

Algorithm pop(): $O(1)$

if isEmpty()

throw a EmptyStackException

$e \leftarrow S[t]$ $\leftarrow$ جيب العنصر من t

$S[t] \leftarrow null$ $\rightarrow$ اصف ال top

$t \leftarrow t-1$ $\rightarrow$ رج t لورا

return e $\rightarrow$ رج العنصر من t

↑ كود ال algorithms هاي بالاجاف موجود بالكتاب (ارجع لـ)

→ implementation stack using an array has a negative aspect

← المشكلة انه لاننا اصد حجم ال array مبقا مبنيا على حجم الذاكرة زياده
 من الـ N مع انه قد يكوننا احتاج حجم ابر من الـ N جزءه
 اي انه بحجز space usage $O(N)$ حيث N هي عدد عناصر ال array بنا انا احتاج
 فقط $O(n)$ حيث n هي عدد عناصر ال stack في الـ

(*) Implemented a stack with a Generic Linked List

(*) Generic $\rightarrow$ يعني بقدر اخذت اكثر من data type مختلف

(*) مثل ما حطينا عليه الاضافة والحذف بتكون من عند top لل stack اي من نفس المكان

لـ من افضل $\rightarrow$ اضيف و اصف من عند head و $tail$ من عند $tail$

لـ من افضل / ان يكون head هو ال top لل stack

منه الاضافة والحذف من عند $O(1)$ بتكون عليه الحذف من $tail$ $O(n)$

(*) كود الجافا لل stack بـ استخدام Generic Linked List



```

public class NodeStack <E> Generic implements stack <E> {
    protected Node <E> top; ← head node
    protected size int size;

```

```

    public NodeStack() {
        top = null;
        size = 0;
    }

```

Constructor

```

    public int size() {
        return size;
    }

```

```

    public boolean isEmpty() {
        if (top == null) return true; → Empty
        return false; } → not Empty

```

```

    public void push(E elem) {
        Node <E> v = new Node <E> (elem, top); ← new node to be inserted
        top = v; ← head ← العنصر الجديد
        size++;
    }

```

element next reference

```

    public E top() throw EmptyStackException {
        if (isEmpty()) throw new EmptyStackException("stack is Empty");
        return top.getElement(); } ← مرجع العنصر في top

```

```

    public E pop() throw EmptyStackException {
        if (isEmpty()) throw new EmptyStackException("stack is Empty");
        E temp = top.getElement(); ← خزنت العنصر في top قبل ما احذفه
        top = top.getNext(); ← من ان top حقة (خزنت)
        size--;
        return temp; } ← مرجع العنصر في top

```

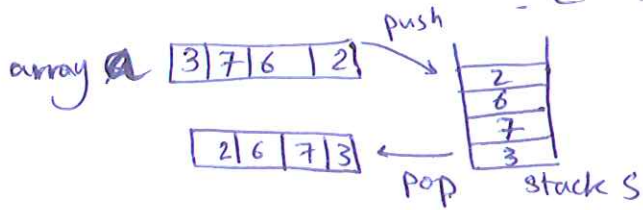
```

}

```

* Reversing an Array Using a stack

الفترة التي نعمل push من خلالها array إلى stack بالترتيب في ال stack ثم بعد ذلك pop من stack إلى array بالترتيب عكس ما كان في array



```
public static <E> void reverse( E[] a) {
    stack<E> S = new ArrayStack<E>(a.length);
    for (int i=0; i<a.length; i++)
        S.push(a[i]);
    for (int i=0; i<a.length; i++)
        a[i] = S.pop();
}
```

5.2 Queues collection of objects that are inserted and removed according to the FIFO principle (First-in First-out)

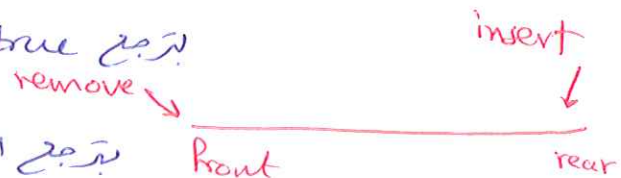
← أول obj يدخل Queue هو أول obj يخرج

* The Queue Abstract Data Type (ADT)

first element in sequence Front of Queue أول عناصر ال queue

end element in sequence rear آخر عناصر ال queue

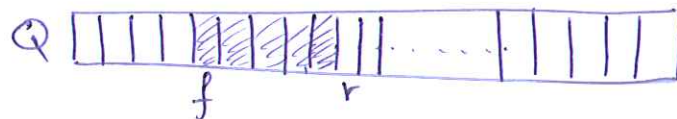
- [1] enqueue(e) :- queue و rear ال element يضاف
- [2] dequeue() :- يرجع العنصر ال Front و يضاف ال Front
- [3] size() :- يرجع عدد العناصر في ال queue
- [4] isEmpty() :- يرجع true إذا كان فارغاً و false إذا لم يكن
- [5] front() :- يرجع العنصر ال Front في ال queue



operation	output	front ~~Queue~~ → rear
enqueue(5)	-	(5)
enqueue(3)	-	(5, 3)
dequeue()	<u>5</u>	(3)
enqueue(7)	-	(3, 7)
dequeue()	3	(7)
front()	7	(7)
isEmpty()	false	(7)
size()	1	(7)

* A Simple Array-Based Queue Implementation

Queue's elem. is in array Q at index f ← r is next var. of index r ← array Q's next var. of index r ←



array-out-of-bound error ←

Q[0] → Q[N-1] no circular array
 $(f+1) \bmod N$ ← r if f → array's next var. of index r ←
 $(r+1) \bmod N$ ←

Algorithm size():
 return $(N - f + r) \bmod N$

Algorithm isEmpty():
 return $(f = r)$

Algorithm front():
 if isEmpty() then
 throw EmptyQueueException
 return Q[f]

Algorithm dequeue():
 if isEmpty() then
 throw EmptyQueueException
 tmp ← Q[f]
 Q[f] ← null
 f ← $(f+1) \bmod N$
 return tmp

Algorithm enqueue(e):
 if size() = N-1 then
 throw FullQueueException
 Q[r] ← e
 r ← $(r+1) \bmod N$

5

⊛ Implementing a Queue with a Generic Linked List

← Queue implementing → Generic Linked list

عدده آخری لا پڑتا تو عمل علی مبدأ FIFO یا ای لہ حیفۃ سے tail رخ ا حذف سے head دیکھیں

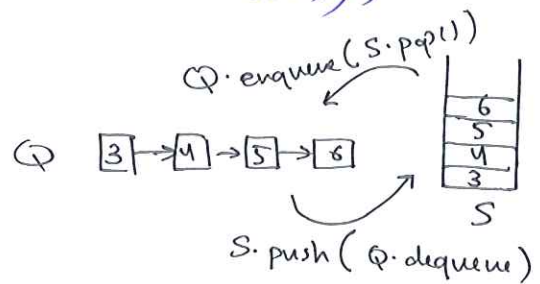
۱۱. فرض کنید head و حقیقت در tail، علاوه بر کلمه $O(1)$ و ظرف $O(n)$

* Reversing a Queue using a stack

```

public static void reverse( Queue <E> Q ) {
    stack <E> S = new ArrayStack<E> ( Q.size() );
    while ( ! Q.isEmpty() )
        S.push( Q.dequeue() );
    while ( ! S.isEmpty() )
        Q.enqueue( S.pop() );
}

```



الانقلاب ببطء وبعيد اني افضي عناصر ال Queue في stack ثم اطلع العناصر من stack واضيقها
 ان كان ال Queue من نوع LIFO و بالانقلاب FIFO حيا يتبع ال reverse ببطء

5.3 Double-Ended Queues (Deque)

The deque abstract data type is richer than both the stack and the Queue ADTs, pronounced "deck"

* The Deque Abstract Data Type (ADT)

- [1] addFirst(e) :-ضيف elem في ان head ال deque (بالاول)
- [2] addLast(e) :-ضيف ~ ~ ~ tail ~ (بالآخر)
- [3] removeFirst() :-يُحذف head ال deque ويرجع العنصر الـ first
- [4] removeLast() :- ~ ~ ~ ~ tail ~
- [5] getFirst() :-يرجع العنصر الـ first في head
- [6] getLast() :-tail ~ ~ ~ ~
- [7] size() :-يرجع عدد العناصر في deque
- [8] isEmpty() :-true لو فارغ false لو مش فارغ

لنفسه، بال head و ال tail هو ابدأ و ينتهي ال DLL في ال node
 ال header و ال header ال tail في DLL هو عبارة عن dummy node
 لا يوجد عليه

operation	output	Deque
addFirst(3)	-	(3)
addFirst(5)	-	(<u>5</u> , 3)
removeFirst()	5	(3)
addLast(7)	-	(3, <u>7</u>)
removeFirst()	3	(7)
addFirst(9)	-	(<u>9</u> , 7)
getLast()	7	(9, 7)
getFirst()	9	(9, 7)
isEmpty()	false	(9, 7)
size()	2	(9, 7)

* Implementing a Deque

الافضل استخدام DLL بدلاً من SLL لأنه، للضافة والحذف يتغير موضع رأس
الصفحة وصله جعل $O(1)$

```
public class NodeDeque<E> implements Deque<E> {
    protected DLNode<E> header, trailer;
    protected int size;
```

Constructor

```
public NodeDeque() {
    header = new DLNode<E>();
    trailer = new DLNode<E>();
    header.setNext(trailer);
    trailer.setPREV(header);
    size = 0;
}
```

```
public int size() {
    return size; }

```

```
public boolean isEmpty() {
    if (size == 0) return true; → empty
    return false; } → not Empty

```

```
public E getFirst() throws EmptyDequeException {
    if (isEmpty())
        throw new EmptyDequeException ("Deque is empty");
    return header.getNext().getElement(); → العنصر الأول في الDeque هو header
}

```

```
public E getLast() throws EmptyDequeException {
    if (isEmpty())
        throw new EmptyDequeException ("Deque is empty");
    return trailer.getPrev().getElement(); → العنصر الأخير في الDeque هو trailer
}

```

```
public void addFirst(E o) {
    DLNode <E> second = header.getNext(); ← تسمية لـ header
    DLNode <E> First = new DLNode <E> (o, header, second); ← العنصر الجديد مع مؤشراته
    second.setPrev(First);
    header.setNext(First); ← مؤشرات الربط
    size++; }

```

```
public void addLast(E o) {
    DLNode <E> second = trailer.getPrev(); ← تسمية لـ trailer
    DLNode <E> First = new DLNode <E> (o, second, trailer); ← العنصر الجديد مع مؤشراته
    second.setNext(First);
    trailer.setPrev(First); ← مؤشرات الربط
    size++; }

```

```

public E removeLast() throws EmptyDequeException {
    if (isEmpty())
        throw new EmptyDequeException("Deque is empty");
    DLNode <E> Last = trailer.getPrev(); ← اوصول للآخر اللى بيدي
    E o = Last.getElement(); ← اطلع العنصر اللى بيدي
    DLNode <E> secondtoLast = last.getPrev(); ← تسمية للآخر اللى قبل
    trailer.setPrev(secondtoLast); ← اللى بيدي احذف
    secondtoLast.setNext(trailer); ← مفصلات
    size --;
    return o;
}

```

```

public E removeFirst() throws EmptyDequeException {
    if (isEmpty())
        throw new EmptyDequeException("Deque is empty");
    DLNode <E> first = header.getNext(); ← اوصول للآخر اللى بيدي احذف
    E o = first.getElement(); ← اطلع العنصر اللى بيدي
    DLNode <E> firstIn = first.getNext(); ← تسمية للآخر اللى بعد اللى
    header.setNext(firstIn); ← بيدي احذف
    firstIn.setPrev(header); ← مفصلات
    size --;
    return o;
}

```

Capacity limited
(*) Implementing ~~to~~ Queue using the deque ADT methods

```
→ public boolean isEmpty() {  
    return deque.isEmpty();  
}  
→ public int size() {  
    return deque.size();  
}  
→ public boolean enqueue(Element e) {  
    if (capacity == deque.size())  
        return false;  
    deque.addLast(e);  
    return true;  
}  
→ public Element dequeue() {  
    if (deque.isEmpty())  
        return null;  
    return deque.removeFirst();  
}  
→ public Element front() {  
    if (deque.isEmpty())  
        return null;  
    return deque.getFirst();  
}
```

(*) Implementing stack using deque

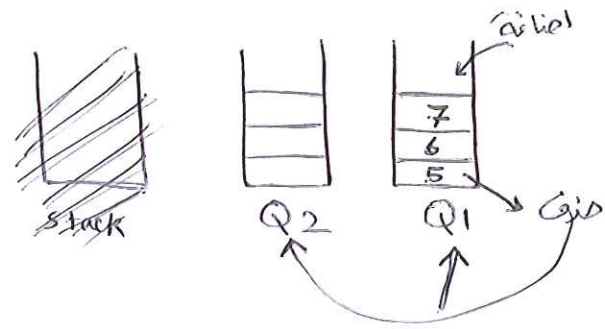
push(e) { d.addFirst(e); }		pop() { return d.removeFirst(); }		top() { return d.getFirst(); }
----------------------------------	--	---	--	--------------------------------------

⊛ Implementing the stack ADT using 2 Queues



```
→ push (object o) {
    Q1.enqueue(o)
}
```

```
→ object pop() {
    while (Q1.size != 1)
        Q2.enqueue(Q1.dequeue());
```



کے اعتبار ان کے عنصر stack

```
    object Tmp = Q1.dequeue();
    while (!Q2.isEmpty())
        Q1.enqueue(Q2.dequeue());
    return Tmp;
}
```

⊛ pop() کے اصل اگلے dequeue کے لئے محفوظ

Q2 کے لئے ماؤصل کے آخر عنصر
الے ہوئے آخریں اضافہ دھروں
نہر کے تیزفے میں ال stack
فہو الے کے اعلیٰ تmp الے الے الے
نہر برجے الے الے الے الے الے

```
→ object top() {
    object Tmp = pop();
    push(T);
    return T;
}
```

⊛ top() ← سب سے الے الے الے الے الے

```
→ int size() {
    return Q.size();
}
```

نہر الے الے الے الے الے
نہر الے الے الے الے الے

```
→ boolean isEmpty() {
    return Q1.isEmpty();
}
```