Deques and Randomized Queues

算法

Dequeue

双端队列，栈和队列的泛化提供从数据结构的头部和尾部增添及删除元素。

```java
import edu.princeton.cs.algs4.StdIn;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Iterator;
import java.util.NoSuchElementException;

public class Deque<Item> implements Iterable<Item> {
    private Node first;
    private Node last;
    private int N;

    private class Node {
        Item item;
        Node previous;
        Node next;
    }

    public Deque() {
        first = null;
        last = null;
        N = 0;
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public int size() {
        return N;
    }

    public void addFirst(Item item) {
        if (item == null) throw new IllegalArgumentException("item
is null...");
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        if (isEmpty()) {
            first = last;
            first.next = null;
        } else {
            first.next = oldfirst;
            oldfirst.previous = first;
        }
        N++;
    }
```

```
    public void addLast(Item item) {
        if (item == null) throw new IllegalArgumentException("item
is null...");
        Node oldlast = last;
        last = new Node();
        last.item = item;
        if (isEmpty()) {
            first = last;
            last.previous = null;
        } else {
            last.previous = oldlast;
            oldlast.next = last;
        }
        N++;
    }

    public Item removeFirst() {
        if (isEmpty()) throw new NoSuchElementException("Stack unde
rflow");
        Item item = first.item;
        first = first.next;
        N--;
        return item;

    }

    public Item removeLast() {
        if (isEmpty()) throw new NoSuchElementException("Stack unde
rflow");
        Item item = last.item;
        last = last.previous;
        N--;
        return item;
    }


    public Iterator<Item> iterator() {
        return new DequeIterator();
    }

    private class DequeIterator implements Iterator<Item> {
        private Node current = first;

        public boolean hasNext() {
            return first != null;
        }

        public Item next() {
            if (!hasNext()) throw new NoSuchElementException("no mo
```

```
re items to return");
            Item item = current.item;
            current = current.next;
            return item;
        }
    }

    public static void main(String[] args) {
        try {
            FileInputStream input = new FileInputStream("/Users/gua
nglinzhou/Dropbox/Algorithms/Chapter1/Deques and Randomized Queue
s/src/tobe.txt");
            System.setIn(input);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        Deque<String> deque = new Deque<>();
        while (!StdIn.isEmpty()) {
            String str = StdIn.readString();
            if (str.equals("-")) {
                deque.removeFirst();
            } else {
                deque.addLast(str);
            }
        }
        System.out.println(deque.size() + " strings left on queu
e.");
    }
}
```

其中，刚开始代码出错的地方为在方法addLast()：

```
last = new Node();
last.item=item;
last.previous=oldlast;
oldlast.next=last;
N++;
if(N==1)
    first=last;
```

代码出错的原因当添加第一个元素时，oldlast、first均为null，在oldlast.next=last这条语句是会出现NullPointerException异常,因为oldlast本身为null，没有next。

每个方法的复杂度为常数级别，一个Node的字节为48bytes：

Node	Bytes
内存开销	16
内部类的额外开销	8
引用item	8
引用previous	8
引用next	8

Randomized queue

通过一个数组来存放存放队列的元素。
删除队列元素是将一个随机位置的元素与队尾元素交换，然后令队尾元素指向空节点。
迭代器的实现是新建一份数组的拷贝，每调用一次next就删除一个元素，当元素的个数为0时迭代完毕。

```java
import edu.princeton.cs.algs4.StdRandom;
import java.util.Iterator;
public class RandomizedQueue<Item> implements Iterable<Item> {
    private int N;
    private Item[] a;

    public RandomizedQueue() {
        N = 0;
        a = (Item[]) new Object[1];
    }

    public Item[] resize(int max) {
        Item[] temp = (Item[]) new Object[max];
        for (int i = 0; i < N; i++) {
            temp[i] = a[i];
        }
        a = temp;
        return a;
    }

    public boolean isEmpty() {
        return N == 0;
    }

    public int size() {
        return N;
    }

    public void enqueue(Item item) {
        if (N == a.length)
            a = resize(2 * a.length);
        a[N++] = item;

    }

    public Item dequeue() {
        int i = StdRandom.uniform(N);//return a pseudo-random integ
er between 0~N-1
        Item item = a[i];
        a[i] = a[N - 1];
        a[N - 1] = item;
        a[--N] = null;//avoid loitering
        if (N <= a.length / 4)
            a = resize(a.length / 2);
        return item;
    }

    public Item sample() {
```

```
        int i = StdRandom.uniform(N);//return a pseudo-random integ
er between 0~N-1
        return a[i];
    }

    @Override
    public Iterator<Item> iterator() {
        return new RandomizedQueueIterator();
    }

    private class RandomizedQueueIterator implements Iterator<Item>
{
        private int i = N;

        public boolean hasNext() {
            return i != 0;
        }

        public Item next() {
            int j = StdRandom.uniform(i);//return a pseudo-random i
nteger between 0~N-1
            Item item = a[j];
            a[j] = a[i - 1];
            a[i - 1] = item;
            a[--i] = null;
            return item;
        }
    }

    public static void main(String[] args) {

    }
}
```