Project 1 : Bag and Deque (Due date: 06/12/2016)

The purpose of this project is to implement elementary data structures using arrays and linked lists, and to introduce you to generics and iterators.

Problem 1. (Bag) A Bag is an iterable collection that provides clients the ability to collect items. Write a java code that store data from input file input.txt into bag and print it with following API:

method	description
Bag()	constructs an empty bag
boolean isEmpty()	is the bag empty?
int size()	returns the number of items on the bag
void add(Item item)	adds item to the front of the bag
Iterator<Item> iterator()	returns an iterator over items in the bag in order from front to end String toString() returns a string representation of the bag

Corner cases. Throw a java.lang.NullPointerException if the client attempts to add a null item; throw a java.lang.UnsupportedOperationException if the client calls the remove() method in the iterator; throw a java.util.NoSuchElementException if the client calls the next() method in the iterator and there are no more items to return.

```
$ java Bag


size of bag = 3
are
how
hi
```

Problem 2. (Deque) A double-ended queue or deque (pronounced \deck") is a generalization of a stack and a queue that supports adding and removing items from either the front or the back of the data structure. Create a generic iterable data type LinkedDeque<Item> in LinkedDeque.java that uses a linked list to implement the following deque API:

method	description
LinkedDeque()	constructs an empty deque
boolean isEmpty()	is the deque empty?
int size()	returns the number of items on the deque
void addFirst(Item item)	adds item to the front of the deque
void addLast(Item item)	adds item to the end of the deque
Item removeFirst()	removes and returns the item from the front of the deque
Item removeLast()	removes and returns the item from the end of the deque
Iterator<Item> iterator()	returns an iterator over items in the deque in order from front to end
String toString()	returns a string representation of the deque

Corner cases. Throw a java.lang.NullPointerException if the client attempts to add a null item;
throw a java.util.NoSuchElementException if the client attempts to remove an item from an empty deque; throw a java.lang.UnsupportedOperationException if the client calls the remove() method in the iterator; throw a java.util.NoSuchElementException if the client calls the next() method in the iterator and there are no more items to return.

Performance requirements. Your deque implementation must support each deque operation (including construction) in constant worst-case time and use space proportional to linear in the number of items currently in the deque. Additionally, your iterator implementation must support each operation (including construction) in constant worst-case time.

```
$ java Deque


false
(364 characters ) There is grandeur in this view of life , with its several powers , having been originally
breathed into a few forms or into one ; and that , whilst this planet has gone cycling on according to
the fixed law of gravity , from so simple a beginning endless forms most beautiful and most wonderful
have been , and are being , evolved . ~ Charles Darwin , The Origin of Species
true
```