```
package linkedCollections;

import java.util.AbstractCollection;
import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 * Implementation of the Collection interface based on a
 * null-terminated, doubly-linked list with no dummy node.
 * Note this collection is not ordered as a list
 * since new elements are always added at the beginning.
 */
public class DoublyLinkedCollection<E> extends AbstractCollection<E> {
  private Node head = null;
  private int size = 0;

  @Override
  public boolean add(E item) {
    // add at beginning
    Node temp = new Node(item, head, null);
    if (head != null)
{
      head.previous = temp;
    }

    head = temp;
    ++size;
    return true;
  }


  @Override
  public Iterator<E> iterator() {
    return new LinkedIterator();
  }

  @Override
  public int size() {
    return size;
  }

  /**
   * Node type for this linked list.
   */
  private class Node {
    public E data;
    public Node next;
    public Node previous;

    public Node(E data, Node next, Node previous) {
      this.data = data;
      this.next = next;
      this.previous = previous;
    }
  }

  /**
   * Iterator for this linked list.
   */
  private class LinkedIterator implements Iterator<E> {
    // Class invariants:
    // 1) cursor points to the next element to be returned by next()
    //    (null if the list is empty or there are no more elements)
    // 2) pending points to the element just returned by next(), null value
    //    indicates that remove() may not be called

    private Node cursor = head;
    private Node pending = null;

    @Override
    public boolean hasNext() {
      return cursor != null;
    }

    @Override
    public E next() {
      if (!hasNext()) throw new NoSuchElementException();

      pending = cursor;
      cursor = cursor.next;
      return pending.data;
    }

    @Override
    public void remove() {
      if (pending == null) throw new IllegalStateException();

      // unlink pending node
      if (pending.previous != null)
      {
        pending.previous.next = pending.next;
      }

      if (pending.next != null)
      {
        pending.next.previous = pending.previous;
      }

      // if we're deleting the head, update head reference
      if (pending == head)
      {
        head = pending.next;
      }

      --size;
      pending = null;
    }
  }
}
```