**Задача 1.**

Написать итератор по двумерному массиву. (Отдельный класс. Наследуемся от Iterator<T>)

Протестировать в main.

**Задача 2.**

Создать список оценок учеников с помощью ArryList, заполнить случайными

оценками. Найти самую высокую оценку с использованием итератора.

**Задача 3.**

Можно писать все в main. Не нужно создавать новых классов.

Имеется текст. Следует составить для него частотный словарь:

Создать Map<Character, Integer> для символов, в который мы заносим символ и его количество.

**Задача 4.**

Создать свое исключение CustomException.

Попробовать добавить в массив int на 8 элементов 10 чисел.

Обернуть в try/catch, словить ArrayIndexOutOfBoundsException и вывести на экран сообщение:

"Array is to small, expand the array". После чего выкинуть свое исключение **CustomException.**

**Задача 5.**

Дан указатель на последний элемент двусвязного списка(head). Head может быть null, это говорит о том, что список пуст, возвращаем null. Поменять порядок элементов в списке на обратный. Измените следующий(next) и предыдущий (prev) указатели всех узлов, чтобы направление списка изменилось. Вернуть список на последний элемент измененного(перевернутого) списка.

**Описание:**

Закончите функцию ниже, которая изменит порядок списка на обратный. Функция принимает следующие параметры:

head – указатель на последний элемент списка в DoublyLinkedList

**Формат ввода:**

Первая линия принимает integer t – количество элементов в списке

Следующие n линий – представляют собой элементы двусвязного списка.

**Ограничения:**

0 <= n <= 1000

0 <= DoubleLinkedListNode.data <= 1000

**Формат вывода:**

Должен вернуться указатель на последний элемент перевернутого списка.

**Пример ввода:**

4

1

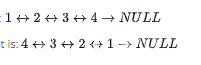
2

3

4

**Пример вывода:**

**4 3 2 1**



**package** by.academy.lesson10;

**import** java.io.IOException;

**import** java.util.Scanner;

**public** **class** Solution {

**static** **class** DoublyLinkedListNode {

**public** **int** data;

**public** DoublyLinkedListNode next;

**public** DoublyLinkedListNode prev;

**public** DoublyLinkedListNode(**int** nodeData) {

**this**.data = nodeData;

**this**.next = **null**;

**this**.prev = **null**;

}

}

**static** **class** DoublyLinkedList {

**public** DoublyLinkedListNode head;

**public** DoublyLinkedListNode tail;

**public** DoublyLinkedList() {

**this**.head = **null**;

**this**.tail = **null**;

}

**public** **void** insertNode(**int** nodeData) {

DoublyLinkedListNode node = **new** DoublyLinkedListNode(nodeData);

**if** (**this**.head == **null**) {

**this**.head = node;

} **else** {

**this**.tail.next = node;

node.prev = **this**.tail;

}

**this**.tail = node;

}

}

**public** **static** **void** printDoublyLinkedList(DoublyLinkedListNode node, String sep) **throws** IOException {

**while** (node != **null**) {

System.***out***.print(String.*valueOf*(node.data));

node = node.next;

**if** (node != **null**) {

System.***out***.print(sep);

}

}

}

// Complete the reverse function below.

/\*

\* For your reference:

\*

\* DoublyLinkedListNode { int data; DoublyLinkedListNode next;

\* DoublyLinkedListNode prev; }

\*

\*/

**public** **static** DoublyLinkedListNode reverse(DoublyLinkedListNode head) {

**return** null;

}

**private** **static** **final** Scanner ***scanner*** = **new** Scanner(System.***in***);

**public** **static** **void** main(String[] args) **throws** IOException {

DoublyLinkedList llist = **new** DoublyLinkedList();

**int** llistCount = ***scanner***.nextInt();

***scanner***.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

**for** (**int** i = 0; i < llistCount; i++) {

**int** llistItem = ***scanner***.nextInt();

***scanner***.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

llist.insertNode(llistItem);

}

DoublyLinkedListNode llist1 = *reverse*(llist.head);

*printDoublyLinkedList*(llist1, " ");

***scanner***.close();

}

}