

Part 1

[1.1. Создать интерфейс MyList](#)

[1.2. Создать класс MyListImpl, который реализует MyList](#)

[1.3. Переопределить метод toString](#)

[1.4. Продемонстрировать работу всех методов контейнера](#)

Part 2

[2.1. Добавить к интерфейсу MyList наследование интерфейса Iterable<Object>](#)

[2.2. Продемонстрировать работу итератора](#)

Part 3

[3.1. Определить интерфейс ListIterator](#)

[3.2. Создать интерфейс ListIterable](#)

[3.3. Добавить к классу MyListImpl реализацию интерфейса ListIterable](#)

[3.4. Добавить в класс MyListImpl метод ListIterator](#)

[3.5. Продемонстрировать работу итератора ListIterator](#)

Класс Demo

Проект должен называться Practice2.

Корневой пакет для всех классов - ua.nure.your_last_name.Practice2

В корневом пакете должен располагаться класс Demo, который демонстрирует работу всех частей.

Содержимое класса Demo должно быть таким каким оно дано в самом конце документа.

Скопировать и вставить как есть.

В комментариях - информация, какой вывод должен у вас быть.

Перед заливкой в репозиторий убедиться, что приложение генерирует правильный результат.

Также перед заливкой удалить комментарии (Sonar определяет trailing комментарий как minor issue).

Part 1

1.1. Создать интерфейс **MyList**

```
public interface MyList {  
    // appends the specified element to the end of this list  
    void add(Object e);  
  
    // removes all of the elements from this list  
    void clear();  
  
    // removes the first occurrence of the specified element  
    // from this list  
    boolean remove(Object o);  
  
    // returns an array containing all of the elements  
    // in this list in proper sequence  
    Object[] toArray();  
  
    // returns the number of elements in this list  
    int size();  
  
    // returns true if this list contains the specified element  
    boolean contains(Object o);  
  
    // returns true if this list contains all of the elements  
    // of the specified list  
    boolean containsAll(MyList c);  
}
```

1.2. Создать класс **MyListImpl**, который реал **изует** **MyList**

1.3. Переопределить метод toString

Переопределить метод toString таким образом, чтобы результат выводился в виде:

[результат вызова toString для элемента 1, результат вызова toString для элемента 2, ...]

1.4. Продемонстрировать работу всех методов контейнера

См. код класса Demo.

Part 2

2.1. Добавить к интерфейсу **MyList** наследование интерфейса **Iterable<Object>**

(интерфейс `java.lang.Iterable`)

```
MyList extends Iterable<Object>
```

Реализовать в контейнере **MyListImpl** метод

```
public Iterator<Object> iterator() {  
    return new IteratorImpl();  
}
```

который возвращает объект внутреннего класса **IteratorImpl**:

```
private class IteratorImpl implements Iterator<Object> {  
  
    // returns true if the iteration has more elements  
    public boolean hasNext() {  
        // ...  
    }  
  
    // returns the next element in the iteration  
    public Object next() {  
        // ...  
    }  
  
    // removes from the underlying collection the last element  
    // returned by this iterator  
    public void remove() {  
        // ...  
    }  
}
```

Алгоритм в методе `remove` может быть следующим:

ЕСЛИ ПЕРЕД ВЫЗОВОМ `remove` НЕ БЫЛ ВЫЗВАН МЕТОД `next`
ИЛИ ПЕРЕД ВЫЗОВОМ `remove` БЫЛ ВЫЗВАН `remove` (повторный вызов `remove`)
ТО ВЫБРОСИТЬ ИСКЛЮЧЕНИЕ (*так и вставить в код*):
`throw new IllegalStateException();`
В ДАННОМ МЕСТЕ ОПРЕДЕЛИТЬ И УДАЛИТЬ СООТВЕТСТВУЮЩИЙ ЭЛЕМЕНТ

2.2. Продемонстрировать работу итератора

См. код класса `Demo`.

Part 3

3.1. Определить интерфейс **ListIterator**

```
// extends java.util.Iterator !!!
interface ListIterator extends Iterator<Object> {

    // returns true if this list iterator has more elements when traversing
    // the list in the reverse direction
    boolean hasPrevious();

    // returns the previous element in the list and moves the cursor
    // position backwards
    Object previous();

    // replaces the last element returned by next or previous with
    // the specified element
    void set(Object e);

    // replaces the last element returned by next or previous with
    // the specified element
    void remove();
}
```

Методы set/remove могут быть вызваны только после next/previous.

Повторный вызов (подряд) set/remove влечет выброс исключения **IllegalStateException** (см. п. 2.1.)

3.2. Создать интерфейс **ListIterable**

```
interface ListIterable {
    ListIterator listIterator();
}
```

3.3. Добавить к классу **MyListImpl** реализацию интерфейса **ListIterable**

```
class MyListImpl implements MyList, ListIterable {...}
```

3.4. Добавить в класс **MyListImpl** метод **ListIterator**

```
public ListIterator listIterator() {
    return new ListIteratorImpl();
}
```

который возвращает объект внутреннего класса **ListIteratorImpl**:

```
private class ListIteratorImpl extends IteratorImpl implements ListIterator {
    // IMPLEMENT ALL METHODS HERE!!!
}
```

3.5. Продемонстрировать работу итератора **ListIterator**

См. код класса Demo.

Класс Demo

```
package ua.nure.your_last_name.Practice2;

import java.util.Iterator;

public class Demo {
    public static void main(String[] args) {
        System.out.println("==== Part1");
        MyList list = new MyListImpl();
        // [A, A2]
        list.add("A");
        list.add("A2");
        System.out.println(list);
        // []
        list.clear();
        System.out.println(list);
        // [A, null, A3]
        list.add("A");
        list.add(null);
        list.add("A2");
        list.add("A3");
        list.remove("A2");
        System.out.println(list);
        // AnullA3
        for (Object el : list.toArray()) {
            System.out.print(el);
        }
        System.out.println();
        // 3
        System.out.println(list.size());
        // false
        System.out.println(list.contains("B"));
        // true
        System.out.println(list.contains("A3"));
        // true
        list.add("A2");
        MyList anotherList = new MyListImpl();
        anotherList.add("A");
        anotherList.add("A2");
        System.out.println(list.containsAll(anotherList));
        // false
        anotherList.add("B");
        System.out.println(list.containsAll(anotherList));
        // true
        list.add("B");
        System.out.println(list.containsAll(anotherList));

        System.out.println("==== Part2");

        list = new MyListImpl();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);

        // 1 2 3 4
        Iterator<Object> it = list.iterator();
        while (it.hasNext()) {
            System.out.print(it.next() + " ");
        }
        System.out.println();
        // [1, 3, 4]
        it = list.iterator();
        it.next();
        it.next();
        it.remove();
        System.out.println(list);

        // 3
        System.out.println(it.next());

        // [1, 4]
        it.remove();
        System.out.println(list);
    }
}
```

```

// class java.lang.IllegalStateException
try {
    it.remove();
} catch (IllegalStateException ex) {
    System.out.println(ex.getClass());
}
System.out.println("==== Part3");

list = new MyListImpl();
list.add(1);
list.add(2);
list.add(3);
list.add(4);

// 1 2 3 4
ListIterator lit = ((ListIterable)list).listIterator();
while (lit.hasNext()) {
    System.out.print(lit.next() + " ");
}
System.out.println();
// 4 3 2 1
while (lit.hasPrevious()) {
    System.out.print(lit.previous() + " ");
}
System.out.println();

list = new MyListImpl();
lit = ((ListIterable)list).listIterator();
// false
System.out.println(lit.hasNext());

// false
System.out.println(lit.hasPrevious());
// Element
list.add("Element");
System.out.println(lit.next());

// false
System.out.println(lit.hasNext());

// true
System.out.println(lit.hasPrevious());
}
}

```