Practice Activities #12

A problem

Implement methods of MyLL (linked List) that extends MyAbstractList. Use **PA12A.java** file, and implement following methods:

1. public void print()
2. public E get(int index)
3. public E getFirst()
4. public E getLast()
5. public void addFirst(E e)
6. public void addLast(E e)
7. public void add(int index, E e)
8. public E removeFirst()
9. public E removeLast()
10. public E remove(int index)
11. public boolean contains(E e)
12. public int indexOf(E e)
13. public int lastIndexOf(E e)
14. public E set(int index, E e)

Correct output:

```
myLL.size(): 11
myLL.isEmpty(): false
[0, 1, 2, 3, 4, 5, 6, 7, 8, 5, 9]
myLL.get(2): 2
myLL.getFirst(): 0
myLL.getLast(): 9
After : add(2, 222):
[0, 1, 222, 3, 4, 5, 6, 7, 8, 5, 9]
After : addFirst(111) and addFirst(999):
[999, 111, 0, 1, 222, 3, 4, 5, 6, 7, 8, 5, 9]
myLL.remove(2): 0
myLL.removeFirst(): 999
myLL.removeLast(): 9
[111, 1, 222, 3, 4, 5, 6, 7, 8, 5]
myLL.contains(111): true
```

```
myLL.contains(5): true
myLL.indexOf(5): 5
myLL.lastIndexOf(5): 9
myLL.lastIndexOf(88): -1
myLL.set(2, 22): 22
myLL.set(1, 3333): 3333
111 3333 22 3 4 5 6 7 8 5
myLL.isEmpty(): true
```

B problem

Implementation of a Stack using Linked List

Instead of using an array, we can also use a linked list to implement a Stack. Create Stack implementation using Linked List structure in Java. In case if stack is empty throw EmptyStackException(); Use **PA12B.java** file, and implement following methods of Stack:

1. public boolean isEmpty()
2. public int size()
3. public void push(E e)
4. public E pop() throws EmptyStackException
5. public E peek() throws EmptyStackException
6. @Override
   public String toString()

implement following methods of StackIterator:

1. @Override
   public boolean hasNext()
2. @Override
   public E next()

Correct output:

```
6_linkedStack.peek(): 6
6_linkedStack.pop(): 6
5_After pop: [ 5 4 3 2 1 ]

5_linkedStack.peek(): 5
5_linkedStack.pop(): 5
4_After pop: [ 4 3 2 1 ]
```

```
4_linkedStack.peek(): 4
4_linkedStack.pop(): 4
3_After pop: [ 3 2 1 ]

3_linkedStack.peek(): 3
3_linkedStack.pop(): 3
2_After pop: [ 2 1 ]

2_linkedStack.peek(): 2
2_linkedStack.pop(): 2
1_After pop: [ 1 ]

1_linkedStack.peek(): 1
1_linkedStack.pop(): 1
0_After pop:

0_After push(25): [ 25 ]
1_After push(26): [ 26 25 ]
2_After push(27): [ 27 26 25 ]
3_After push(28): [ 28 27 26 25 ]
4_After push(29): [ 29 28 27 26 25 ]
5_After push(30): [ 30 29 28 27 26 25 ]
30 29 28 27 26 25
30 29 28 27 26 25
```

C problem

Now you can extend your LinkedStack from B task – with inner StackListIterator class. Implement `hasNext(),next(),hasPrevious(),previous(), nextIndex(),previousIndex()` methods of inner *LinkedStack.StackListIterator class.*

```
class StackListIterator implements ListIterator<E> {
    boolean canRemove = false;
    int previousLoc = -1;
    StackNode<E> current = top;

    @Override
    public boolean hasNext() {
        // your code goes here
        return false;
    }

    @Override
    public E next() {
        // your code goes here
        return null;
    }

    @Override
    public boolean hasPrevious() {
        // your code goes here
        return false;
    }

    @Override
    public E previous() {
        // your code goes here
        return null;
    }

    @Override
    public int nextIndex() {
        // your code goes here
        return -1;
    }
```

```
    @Override
    public int previousIndex() {
        // your code goes here
        return -1;
    }

    @Override
    public void remove() {
        System.err.println("You can access only top element in
stack!");
    }

    @Override
    public void set(E e) {
        System.err.println("You can access only top element in
stack!");
    }

    @Override
    public void add(E e) {
        System.err.println("You can access only top element in
stack!");
    }
}
```

To main add following test:

```
ListIterator<Integer> listIterator =
linkedStack.listIterator();
while (listIterator.hasNext()){
    System.out.print(listIterator.nextIndex() + "_" +
listIterator.next() + " ");
}
System.out.println();


while (listIterator.hasPrevious()){
    System.out.print(listIterator.previousIndex() + "_" +
listIterator.previous() + " ");
}
System.out.println();
```

Correct output:

```
0_30 1_29 2_28 3_27 4_26 5_25
5_25 4_26 3_27 2_28 1_29 0_30
```