HW3

思考构架 SList, SListNode.

```
public class SListNode {
    private Object item;
    private SListNode next;

    SListNode( ){          // 无参构造函数
        item = null;
        next = null;
    }
    SListNode(Object item, SListNode next){   // 带参构造函数
        this.item = item;
        this.next = next;
    }
    SListNode(Object item) {
        this(item, null);
    }
```

```
Public class SList {
    private SListNode head;
    private int size;
    public SList( ) { size = 0; head = null;}
    public boolean isEmpty( ) { … }
    public int length( ) { … }
    public void insertFront(Object obj)
    public void insertEnd(Object obj)
    public Object nth(int position) // 返回第nth的元素.
    public void squish( )
    public void twin( )
```

封装的概念

HW 4

思考构架 DList, DListNode.

```
public class DListNode {
    private Object item;
    private DListNode prev;
    private DListNode next;
public DListNode(Object obj, DListNode p,
                 DListNode n) {
    item = obj;
    prev = prev;
    next = n;
}
}
```

```
public class DList {
    private DListNode head; // 充当 sentinel
    private int size;
    public DListNode newNode(Object obj, p, n) {
        return new DListNode(obj, p, n);
    }
    public boolean isEmpty( ) { … }
    public int length( ) { … }
    DList( ) { head = newNode(obj, null, null);
               head.next = head;
               head.prev = head;
               size = 0;
    }
```

Integer.MIN_VALUE

```
public void insertFront(Object obj);
public void insertBack(Object obj);
public DListNode front( ); 返回第一个元素
public DListNode back( ); 返回最后一个元素
public DListNode next(DListNode node);
public DListNode prev(DListNode node);
public void insertAfter(Object item, DListNode node);
public void insertBefore(Object item, DListNode);
public void remove(DListNode node);
```

If node is null, do nothing

思考如何在引入abstract后一步步构建整个框架

HW5

1. 考虑 ListNode 和 List.

```
public abstract class ListNode {
    protected Object item;
    protected List myList;

    public boolean isValidNode( ){...}
    public Object item( ) throws InvalidNode
                                 Exception
      if( ✓ ) throw new In... ( );
      else return item;
        SetItem( )
```

ListNode 包含基础抽象方法，只声明，不实现！

```
}
```

```
public class InvalidNodeException extends
                                  Exception {
    InvalidNodeException( )
      super( );
    InvalidNodeException (String s)
      { ~~super~~ super(s); }
```

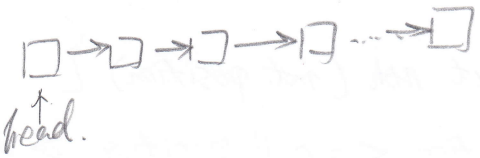
```
public abstract class List {
    protected int size;

    public boolean isEmpty( ){...}
    public int length( ) {...}

    public abstract void insertFront(Object item);
           abstract    insertBack( ... );
           abstract    front( );
           abstract    back( );
           abstract    toString( );
```

抽象方法，需要在子类中具体实现！

2. 如果用 SList，SListNode 怎么实现？

3. 如果用 DList，DListNode 怎么实现？

SList



1. insertFront(Object obj){

如果原先为空. { SListNode node = new SListNode(obj);
head = node; ~~size++;~~
}

否则 { SListNode node = newSList(obj, head); }.
head = node;
size++;
}

⇓ 简化

insertFront(Object obj){

SListNode node = newSList~~Node~~(obj, head);
head = node;
size++;
}

↑ 调用了SListNode ~~构造~~ 函数.

⇒ head = new SListNode(obj, head);

2. insertEnd(Object obj){

if(isEmpty()) { insertFront(obj); }

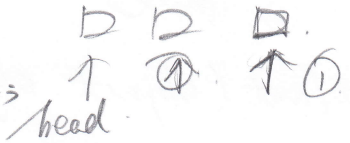
因为没有tail, 所以只能从head往后走.

SListNode currentNode = head;
while(currentNode.next != null){
currentNode = currentNode.next;
}.
currentNode.next = new SListNode(obj);
}
size++;
}

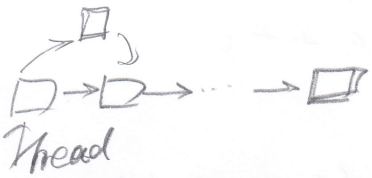
3. public Object nth (int position) {

```
if (position <= 0 || position > length( )) { 예외: ... }
else if (position == 1) { return head.item; }
else { SListNode currentNode = head;
       while (position > 1) {
           currentNode = currentNode.next( );
           position--;
       }
       return currentNode.item;
}
```

position=3



4. public void twin( ) {




```
if (isEmpty( )) return;
SListNode currentNode = head;
while (currentNode.next != null) {
    SListNode node = new SListNode(current.item, currentNode.next);
    currentNode.next = node;
    currentNode = node.next;
}
currentNode.next = new SListNode(current.item);
}
```

DList

1. Public void insertFront(Object obj) {

```
X head = newNode(obj, head, head.next); X 由于head已确定，不能用此语法.
DListNode node = newNode(obj, head, head.next);
head.next = node;
node.next.prev = node;
size++;
}
```

2. public void insertBack(Object obj) 与 insertFront() 类似.

```
DListNode node = newNode(obj, head.prev, head);
head.prev.next = node;
head.prev = node;
size++;
```

3. public DListNode front() 与 public DListNode back() 需要考虑只有head的情况.

```
if(isEmpty()) ...
else return head.next;
```

```
if(isEmpty()) ...
else return head.prev;
```

4. public DListNode next(DListNode node)

```
if(node==null || node.#=back()) return null;
else return node.next;
```

5. public DListNode prev(DListNode node)

```
if(node==null || node==front()) return null;
else return node.prev;
```

6. public void insertAfter(Object item, DListNode node)

```
if(node==null) return;
DListNode nodeAfter = newNode(item, node, node.next);
```

7. public void insertBefore(Object item, DListNode node)

```
if(node==null) return;
DListNode nodeBefore = newNode(item, node.prev, node);
```

HW5 summary

List.java 和 ListNode.java 都是 abstract class. 不允许创建该类对象.

```
public abstract class List {
    protected int size;
    public boolean isEmpty( ) { return size == 0; }
    public int length( ) { return size; }
    public abstract void insertFront (Object item);
    public abstract void insertBack(Object item);
    public abstract ListNode front( );
    public abstract ListNode back( );
    public abstract String toString( );
}
```

抽象方法不要 implementation code!

```
public abstract class ListNode {
    protected Object item;
    protected List myList;
    public boolean isValidNode( ) { return myList != null; }
    public Object item( ) throws InvalidNodeException {
        if (!isValidNode( )) { throw new InvalidNodeException( ); }
        return item;
    }
    public void setItem(Object item) throws InvalidNodeException {
        if (!isValidNode( )) { throw new InvalidNodeException( ); }
        return this.item = item;
    }
    public abstract ListNode next( ) throws InvalidNodeException;
    public abstract ListNode prev( ) throws InvalidNodeException;
    public abstract void insertAfter(Object item) throws InvalidNodeException;
    public abstract void insertBefore(Object item) throws InvalidNodeException;
    public abstract void remove( ) throws InvalidNodeException;
}
```

// 每一个 ListNode 知道它是属于什么 List. ⇒ 正是因为此 instance variable, 那么 List 的一些方法可以移到 ListNode 这儿来!

```
public class InvalidNodeException extends Exception {
    protected InvalidNodeException( ) { super( ); }
    protected InvalidNodeException(String s) { super(s); }
}
```

String describing the Exception

SListNode.java

```
public class SListNode extends ListNode {
  protected SListNode next;
  SListNode(Object i, SList l, SListNode n) {
      item = i;
    myList = l;
      next = n;
  }
public ListNode next() throws InvalidNodeException {
    if (!isValidNode()) {
        throw new InvalidNodeException("next() called on invalid node");
    }
    if (next == null) {
      SListNode node = ((SList) myList).newNode(null, null); // Create a invalid node.
      node.myList = null;
      return node;
    }else{
      return next;
    }
}
public ListNode prev() throws InvalidNodeException {
    if (!isValidNode()) {
        throw new InvalidNodeException("prev() called on invalid node");
    }
    SListNode prev = ((SList) myList).head;          // 把head给prev.
    if (prev == this) {
      prev = ((SList) myList).newNode(null, null);
      prev.myList = null;
    }else{
      while (prev.next != this) {
          prev = prev.next;
      }
    }
    return prev;
}
```

```
public void insertAfter(Object item) throws InvalidNodeException {
    if (!isValidNode()) {
        throw new InvalidNodeException("insertAfter() called on invalid node");
    }
    SListNode newNode = ((SList) myList).newNode(item, next);
    if (next == null) {
        ((SList) myList).tail = newNode;
    }
    next = newNode;
    myList.size++;
}
public void insertBefore(Object item) throws InvalidNodeException {
    if (!isValidNode()) {
        throw new InvalidNodeException("insertBefore() called on invalid node");
    }
    SListNode newNode = ((SList) myList).newNode(item, this);
    if (this == ((SList) myList).head) {
        ((SList) myList).head = newNode;
    } else {
        SListNode prev = (SListNode) prev();
        prev.next = newNode;
    }
    myList.size++;
}
public void remove() throws InvalidNodeException {
    if (!isValidNode()) {
        throw new InvalidNodeException("remove() called on invalid node");
    }
    if (this == ((SList) myList).head) {
        ((SList) myList).head = next;
        if (next == null) {
            ((SList) myList).tail = null;
        }
    } else {
        SListNode prev = (SListNode) prev();
        prev.next = next;
        if (next == null) { ((SList) myList).tail = prev; }
    }
```

```
    myList.size--;
    myList = null;      Make this node an invalid node.
    next = null;        // set other reference to null to improve garbage collection.
  }
} // SList 結束符

// SList.java
public class SList extends List {
  protected SListNode head;
  protected SListNode tail;
  protected SListNode newNode(Object item, SListNode next) {
      return new SListNode(item, this, next);
  }
  public SList() {
    head = null;
    tail = null;
    size = 0;
  }
  public void insertFront(Object item) {
    head = newNode(item, head);
    if(size == 0) { tail = head; }
    size++;
  }
  public void insertBack(Object item) {
    if (head == null) {
      head = newNode(item, null);
      tail = head;
    } else {
      tail.next = newNode(item, null);
      tail = tail.next;
    }
    size++;
  }
  public ListNode front() {
    if(head == null) {
      SListNode node = newNode(null, null);
      node.myList = null;
      return node;
    } else { return head; }
  }
  public ListNode back() {
    if(tail == null) {
      SListNode node = newNode(null, null);
      node.myList = null;
      return node;
    } else { return tail; }
  }
  public String toString() {
    String result = "[ ";
    SListNode current = head;
    while (current != null) {
      result = result + current.item + " ";
      current = current.next;
    }
    return result + "]";
  }
  省略 public static void main(String[] argv) {
  ... }
```

DListNode.java

```
public class DListNode extends ListNode {

    protected DListNode prev;
    protected DListNode next;
    DListNode(Object i, DList l, DListNode p, DListNode n) {
      item = i;
      myList = l;
      prev = p;
      next = n;
    }
    public boolean isValidNode() { return myList != null; }
    public ListNode next() throws InvalidNodeException {
        if(!isValidNode()) {
            throw new InvalidNodeException("next() called on invalid Node.");
        }
        return next;
    }
    public ListNode prev() throws InvalidNodeException {
        if(!isValidNode()) {
            throw new InvalidNodeException("prev() called on invalid Node.");
        }
        return prev;
    }
    public void insertAfter(Object item) throws InvalidNodeException {
        if (!isValidNode()) {
            throw new InvalidNodeException("insertAfter() called on invalid node.");
        }
        DListNode nodeAfter = ((DList) myList).newNode(item, (DList) myList, this, next);
        next = nodeAfter;
        nodeAfter.next.prev = nodeAfter;
        myList.size++;
    }
```