实验5

练习1

题目

要求用顺序存储结构实现线性表。对线性表进行若干次插入和删除，对于每一次插入或删除，若成功，则输出线性表，否则输出出错信息。

执行示例：（加下划线部分为输入内容）
是否要对线性表进行插入和删除？（Y/N）
Y
进行插入还是删除？（1--插入，2--删除）
1
输入插入位置：
0
输入插入元素：
8
线性表为（8）
是否要对线性表进行插入和删除？（Y/N）
Y
进行插入还是删除？（1--插入，2--删除）
1
输入插入位置：
0
输入插入元素：
2
线性表为（2,8）
是否要对线性表进行插入和删除？（Y/N）
Y
进行插入还是删除？（1--插入，2--删除）
1
输入插入位置：
3
输入插入元素：
5
插入位置有误
是否要对线性表进行插入和删除？（Y/N）
Y
进行插入还是删除？（1--插入，2--删除）
1
输入插入位置：
2
输入插入元素：
5
线性表为（2,8,5）
是否要对线性表进行插入和删除？（Y/N）
Y

进行插入还是删除？（1--插入，2--删除）
2
输入删除位置：
2
线性表为（2,8）
是否要对线性表进行插入和删除？（Y/N）
Y
进行插入还是删除？（1--插入，2--删除）
2
输入删除位置：
8
删除位置有误
是否要对线性表进行插入和删除？（Y/N）
Y
进行插入还是删除？（1--插入，2--删除）
2
输入删除位置：
0
线性表为（8）
……
是否要对线性表进行插入和删除？（Y/N）
N
对线性表处理完毕

解析

仿照Python中的List类型创建Java数据结构，自动扩展（空间已满时空间扩展为两倍）与收缩列表空间（空间使用不足四分之一时收缩一半）。

代码

List.java

```
public class List {
    private Object[] listElem;
    private int length;

    // 初始化
    public List() {
        length = 0;
        listElem = new Object[1];
    }

    // 清空列表
    public void clear() {
        length = 0;
        listElem = new Object[1];
    }

    // 是否为空
    public boolean isEmpty() {
        return length == 0;
    }
```

```
    // 返回列表长度
    public int length() {
        return length;
    }

    // 返回下标为index的元素
    public Object get(int index) throws Exception {
        if (index < 0) {
            index = length + index;
        }
        if (index < 0 || index >= length) {
            throw new Exception("Invalid index " + index);
        }
        return listElem[index];
    }

    // 增加元素
    public void append(Object x) {
        // 若空间已满，扩展空间（x2）
        if (length == listElem.length) {
            Object[] listElemTmp = listElem;
            listElem = new Object[2 * listElem.length];
            for (int i = 0; i < length; i++) {
                listElem[i] = listElemTmp[i];
            }
        }
        // 增加元素
        listElem[length++] = x;
    }

    // 插入（允许负下标与大于等于列表长度的下标）
    public void insert(int index, Object x) {
        // 处理负下标与大于等于列表长度的下标
        if (index < 0) {
            index = length + index;
            if (index < 0) {
                index = 0;
            }
        }
        if (index > length) {
            index = length;
        }
        // 若空间已满，扩展空间（x2）
        if (length == listElem.length) {
            Object[] listElemTmp = listElem;
            listElem = new Object[2 * listElem.length];
            for (int i = 0; i < index; i++) {
                listElem[i] = listElemTmp[i];
            }
            for (int i = index + 1; i < length + 1; i++) {
                listElem[i] = listElemTmp[i - 1];
            }
        }
        // 插入元素
        length++;
        listElem[index] = x;
    }
```

```
    // 严格插入（不允许负下标与大于等于列表长度的下标）
    public void strictInsert(int index, Object x) throws Exception {
        if (index < 0 || index > length) {
            throw new Exception("invalid index " + index);
        }
        insert(index, x);
    }

    // 删除（元素不存在不抛出异常）
    public void remove(Object x) {
        int i = 0;
        while (i < length && !listElem[i].equals(x)) {
            i++;
        }
        if (i < length) {
            // 删除元素
            length--;
            for (int j = i; j < length; j++) {
                listElem[j] = listElem[j + 1];
            }
            // 若空间使用率不足四分之一，收缩空间（x1/2）
            if (length <= listElem.length / 4) {
                Object[] listElemTmp = listElem;
                listElem = new Object[listElem.length / 2];
                for (i = 0; i < length; i++) {
                    listElem[i] = listElemTmp[i];
                }
            }
        }
    }

    // 严格删除（元素不存在则抛出异常）
    public void strictRemove(Object x) throws Exception {
        int i = 0;
        while (i < length && !listElem[i].equals(x)) {
            i++;
        }
        if (i < length) {
            // 删除元素
            length--;
            for (int j = i; j < length; j++) {
                listElem[j] = listElem[j + 1];
            }
            // 若空间使用率不足四分之一，收缩空间（x1/2）
            if (length <= listElem.length / 4) {
                Object[] listElemTmp = listElem;
                listElem = new Object[listElem.length / 2];
                for (i = 0; i < length; i++) {
                    listElem[i] = listElemTmp[i];
                }
            }
        } else {
            throw new Exception(x + " not in list");
        }
    }

    // 按下标删除（允许负下标与大于等于列表长度的下标）
    public void removeIndex(int index) {
```

```
        // 处理负下标与大于等于列表长度的下标
        if (index < 0) {
            index = length + index;
            if (index < 0) {
                index = 0;
            }
        }
        if (index > length) {
            index = length;
        }
        // 删除元素
        length--;
        for (int i = index; i < length; i++) {
            listElem[i] = listElem[i + 1];
        }
        // 若空间使用率不足四分之一，收缩空间（x1/2）
        if (length <= listElem.length / 4) {
            Object[] listElemTmp = listElem;
            listElem = new Object[listElem.length / 2];
            for (int i = 0; i < length; i++) {
                listElem[i] = listElemTmp[i];
            }
        }
    }

    // 严格按下标删除（不允许负下标与大于等于列表长度的下标）
    public void strictRemoveIndex(int index) throws Exception {
        if (index < 0 || index > length) {
            throw new Exception("invalid index " + index);
        }
        removeIndex(index);
    }

    // 返回列表中首次出现指定元素的下标，若不存在，则返回-1
    public int indexOf(Object x) {
        int i = 0;
        while (i < length && !listElem[i].equals(x)) {
            i++;
        }
        if (i < length) {
            return i;
        } else {
            return -1;
        }
    }

    // 返回列表中首次出现指定元素的下标，若不存在，则抛出异常
    public int strictIndexOf(Object x) throws Exception {
        int i = 0;
        while (i < length && !listElem[i].equals(x)) {
            i++;
        }
        if (i < length) {
            return i;
        } else {
            throw new Exception(x + " is not in list");
        }
    }
```

```
    // 转换为字符串，重写toString方法
    public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append("[");
        if (length > 0) {
            builder.append(listElem[0]);
        }
        for (int i = 1; i < length; i++) {
            builder.append(", ");
            builder.append(listElem[i]);
        }
        builder.append("]");
        return builder.toString();
    }

    // 打印列表
    public void display() {
        System.out.println(toString());
    }
}
```

ListTest.java

```
import java.util.Scanner;

public class ListTest {
    public static void main(String[] args) {
        System.out.println(":exit                          退出程序");
        System.out.println("-c/-clear                      清空列表");
        System.out.println("-d/-disp/-display              打印列表");
        System.out.println("-g/-get i                      打印下标为i的元素");
        System.out.println("-a/-append x                   增加元素x");
        System.out.println("-i/-insert i x                 在下标i处插入元素x");
        System.out.println("-si/-strictInsert i x          在下标i处插入元素x，若下标为
负或大于等于列表长度，则抛出异常");
        System.out.println("-r/-remove x                   删除列表中第一个为x的元素");
        System.out.println("-sr/-strictRemove x            删除列表中第一个为x的元素，若
不存在，则抛出异常");
        System.out.println("-ri/-removeIndex i             删除下标为i的元素");
        System.out.println("-sri/-strictRemoveIndex i      删除下标为i的元素，若不存在，
则抛出异常");
        System.out.println("-io/-indexOf x                 返回列表中首次出现指定元素的下
标，若不存在，则返回-1");
        System.out.println("-sio/-strictIndexOf x          返回列表中首次出现指定元素的下
标，若不存在，则抛出异常");
        System.out.println();

        List lst = new List();
        String in;
        Scanner sc = new Scanner(System.in);
        in = sc.nextLine();
        while (!in.equals(":exit")) {
            int len = in.split(" ").length;
            String operator = in.split(" ")[0];
            if (len == 1 && (operator.equals("-c") || operator.equals("-
clear"))) {
```

```
                lst.clear();
            } else if (len == 1 && (operator.equals("-d") || operator.equals("-
disp") || operator.equals("-display"))) {
                lst.display();
            } else if (len == 2 && (operator.equals("-g") || operator.equals("-
get"))) {
                int index = Integer.parseInt(in.split(" ")[1]);
                try {
                    System.out.println(lst.get(index));
                } catch (Exception e) {
                    System.out.println("Error: " + e.getMessage());
                }
            } else if (len == 2 && (operator.equals("-a") || operator.equals("-
append"))) {
                String x = in.split(" ")[1];
                lst.append(x);
                lst.display();
            } else if (len == 3 && (operator.equals("-i") || operator.equals("-
insert"))) {
                int index = Integer.parseInt(in.split(" ")[1]);
                String x = in.split(" ")[2];
                lst.insert(index, x);
                lst.display();
            } else if (len == 3 && (operator.equals("-si") || operator.equals("-
strictInsert"))) {
                int index = Integer.parseInt(in.split(" ")[1]);
                String x = in.split(" ")[2];
                try {
                    lst.strictInsert(index, x);
                    lst.display();
                } catch (Exception e) {
                    System.out.println("Error: " + e.getMessage());
                }
            } else if (len == 2 && (operator.equals("-r") || operator.equals("-
remove"))) {
                String x = in.split(" ")[1];
                lst.remove(x);
                lst.display();
            } else if (len == 2 && (operator.equals("-sr") || operator.equals("-
strictRemove"))) {
                String x = in.split(" ")[1];
                try {
                    lst.strictRemove(x);
                    lst.display();
                } catch (Exception e) {
                    System.out.println("Error: " + e.getMessage());
                }
            } else if (len == 2 && (operator.equals("-ri") || operator.equals("-
removeIndex"))) {
                int index = Integer.parseInt(in.split(" ")[1]);
                lst.removeIndex(index);
                lst.display();
            } else if (len == 2 && (operator.equals("-sri") ||
operator.equals("-strictRemoveIndex"))) {
                int index = Integer.parseInt(in.split(" ")[1]);
                try {
                    lst.strictRemoveIndex(index);
                    lst.display();
```

```
                } catch (Exception e) {
                    System.out.println("Error: " + e.getMessage());
                }
            } else if (len == 2 && (operator.equals("-io") || operator.equals("-
indexOf"))) {
                String x = in.split(" ")[1];
                System.out.println(lst.indexOf(x));
            } else if (len == 2 && (operator.equals("-sio") ||
operator.equals("-strictIndexOf"))) {
                String x = in.split(" ")[1];
                try {
                    System.out.println(lst.strictIndexOf(x));
                } catch (Exception e) {
                    System.out.println("Error: " + e.getMessage());
                }
            }
            in = sc.nextLine();
        }
        sc.close();
    }
}
```

测试（运行ListTest.java）（输入输出交替）

```
-si 0 8
[8]
-si 0 2
[2, 8]
-si 3 5
Error: invalid index 3
-si 2 5
[2, 8, 5]
-sri 2
[2, 8]
-sri 8
Error: invalid index 8
-sri 0
[8]
:exit
```

心得体会

1. 使用自定义数据结构能很大程度上简化代码，带来便利。
2. 使用数组时，要注意下标异常及空间已满问题。
3. 插入删除操作应当在下标异常时抛出异常，而不是忽略异常。
4. 应当尽可能平衡顺序表的空间利用率与插入删除效率。
5. 数据结构不应带有与其本身无关的方法。