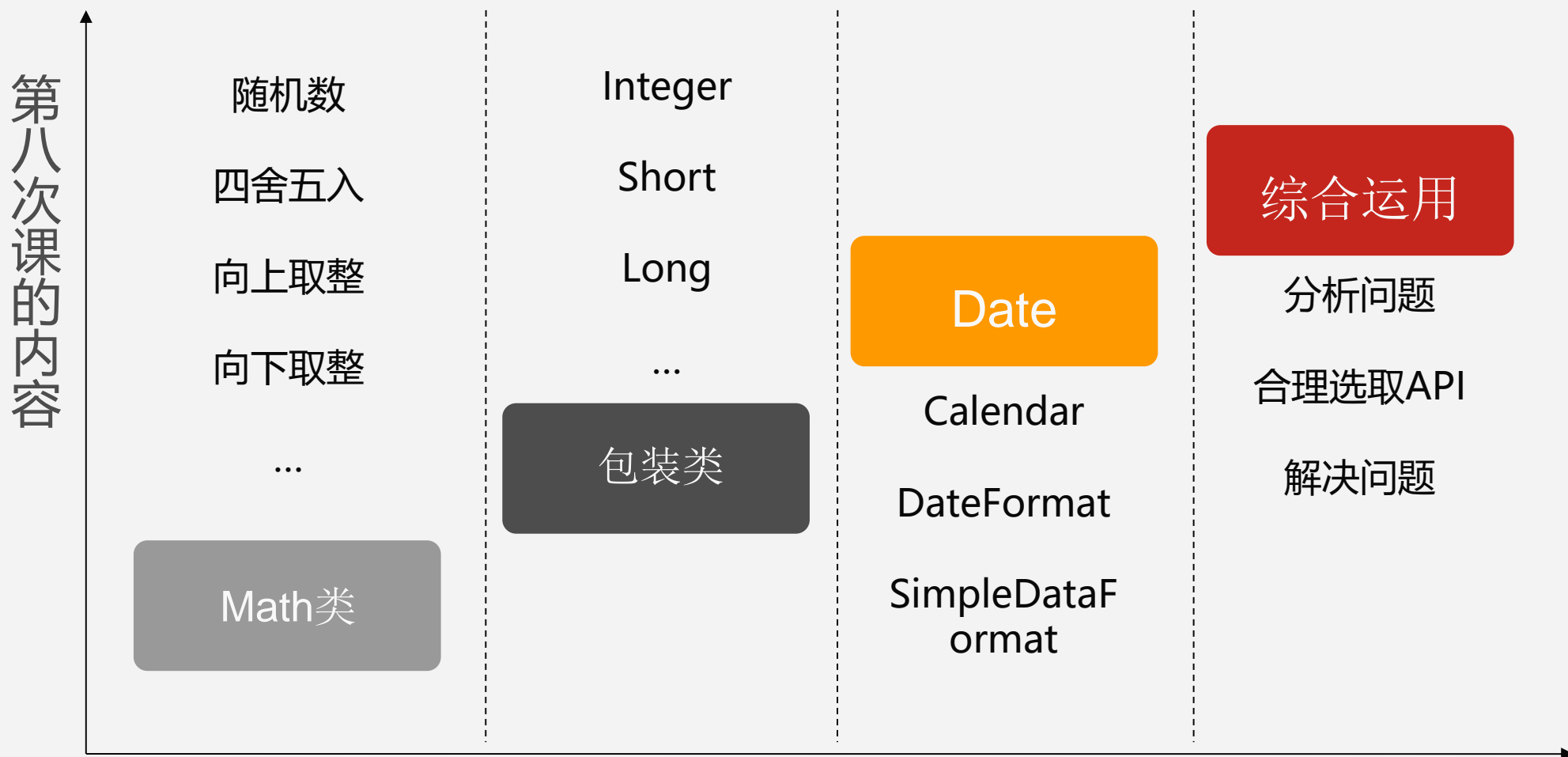




# Java

## 面向对象程序设计

软件学院 贾伟峰



使用集合类，另一个角度模拟一个电信计费系统的部分功能。

## 什么是集合？ 什么是集合类？



13800138000,2017-11-18 09:56:30,2017-11-18 09:58:30,13803721210;15093960507,2017-11-18 09:36:30,2017-11-18 09:56:30,13803723587;.....

把每个字符看成元素的话，这是个字符的集合

13800138000,2017-11-18 09:56:30,2017-11-18 09:58:30,13803721210;15093960507,2017-11-18 09:36:30,2017-11-18 09:56:30,13803723587;.....

把字符串当作元素的话，这又是个字符串的集合。

所以，集合是个抽象的概念。集合类就像容器，用于存储Java类的对象。

这些都“像”容器，注意区别。

## 数组

整数数组  
浮点数数组  
双精度数组  
字符数组  
各种数组

## StringBuffer

和String的区别就是这个存的字符长度不固定，且可动态增删改。访问里面的数据时，还需要toString()

01

02

03

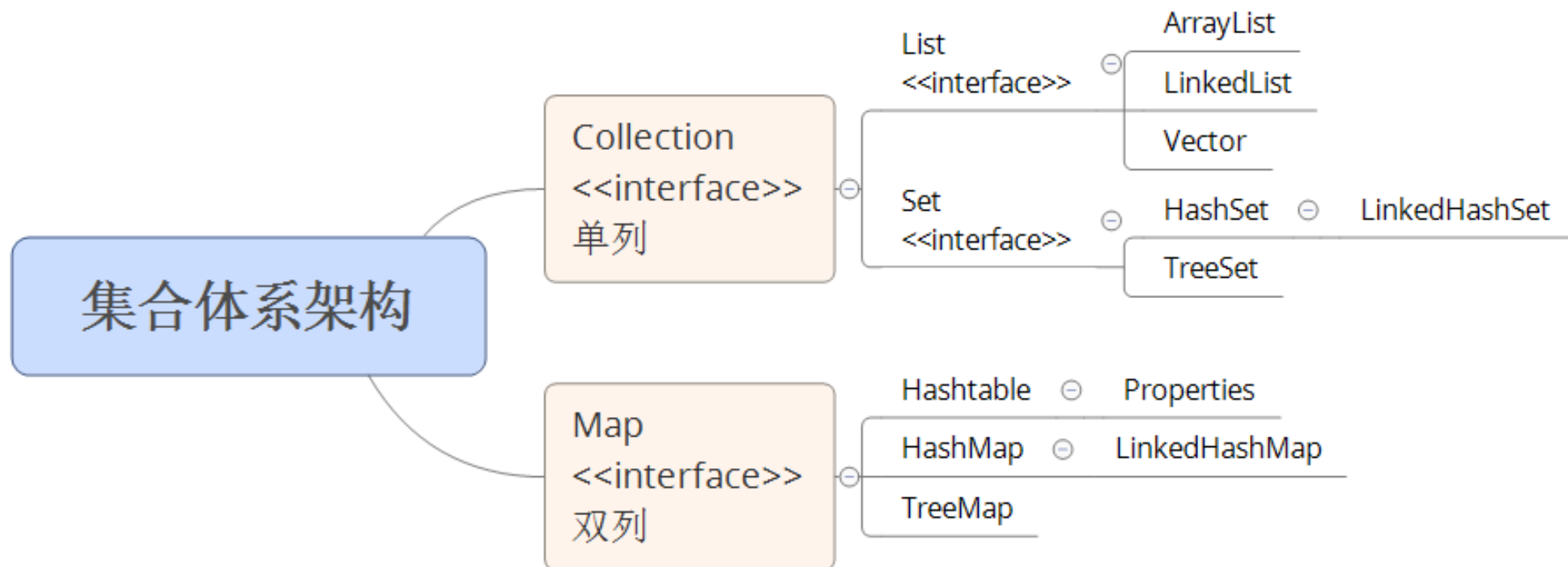
04

## String

可以通过API转成字符数组，字符数组，也“像”容器啊！

## 集合类

可以看成StringBuffer在所有Java对象上的扩展。是可存储任意Java对象的动态数组(或其他结构)。可读可写。功能更强大，更统一。



# 另一个角度模拟电信计费系统

*使用集合类存储通话记录*

之前，是将通话记录存起来，存到StringBuffer中

```
1 import java.util.*;
2 class TelcomUser {
3     private String phoneNumber;
4     private String callTo;
5     private StringBuffer communicationRecords;
6     public TelcomUser(String phoneNumber) {
7         this.phoneNumber = phoneNumber;
8         this.communicationRecords = new StringBuffer();
9     }
10
11     //模拟通话记录的生成
12     void generateCommunicateRecord() {}
13
14     //随机生成被叫号码（后四位随机）并返回
15     private String getCallToPhoneNumber() {}
16
17     //模拟计费办法，以字符串的形式返回保留4位小数的计费结果
18     private String accountFee(long timeStart, long timeEnd) {}
19
20     //打印通话记录
21     void printDetails() {}
22 }
```

假定是被叫号码

假定是主叫号码

假定通话记录存在这里，以分号相隔；每条记录内部有主叫号码、被叫号码、开始时间、结束时间


```
23 //插入通话记录
24 this.communicationRecords.append(this.phoneNumber +
25     " " + timeStart +
26     " " + timeEnd +
27     " " + this.callTo +
28     " ");
```

补充完整吧！



# 使用的时候，先按；分割，然后.....

```
50 //打印通话记录
51 void printDetails() {
52     //获取全部通话记录
53     String allRecords = this.communicationRecords.toString();
54     //分割通话记录
55     String [] recordArray = allRecords.split(";");
56     //循环分割记录内的每一项并输出
57     for(int i = 0; i < recordArray.length; i++) {
58
59     }
60 }
61 }
```



想想循环里面的代码该怎么写？

现在，使用ArrayList存储通话记录

这样，从ArrayList中读出的数据，自然就是一条通话记录...

## 代码就像这个样子

```
1 import java.text.DateFormat;
2 import java.text.SimpleDateFormat;
3 import java.util.*;
4 class TelcomUserBasedOnCollection {
5     private String phoneNumber;
6     private String callTo;
7     private ArrayList communicationRecords;
8     public TelcomUserBasedOnCollection(String phoneNumber) {
9         this.phoneNumber = phoneNumber;
10        this.communicationRecords = new ArrayList();
11    }
12 }
```

注意换了集合类ArrayList

## 代码就像这个样子

```
13 //模拟通话记录的生成
14 void generateCommunicateRecord() {
15     //随机生成通话记录数目
16     int recordNum = new Random().nextInt(10);
17     for(int i = 0; i <= recordNum; i++) {
18         //随机生成第i条通话记录
19         //用Calendar获取当前时间
20         Calendar cal = Calendar.getInstance();
21         //随机减去若干小时(10小时以内)
22         cal.add(Calendar.HOUR, - new Random().nextInt(10));
23         //获得对应毫秒
24         long timeStart = cal.getTimeInMillis();
25         //结束时间开始后的十分钟内随机的一个时间,至少一分钟
26         long timeEnd = timeStart + 60000 + new Random().nextInt(600000);
27
28         //被叫号码
29         this.callTo = getCallToPhoneNumber();
30         //插入通话记录
31         this.communicationRecords.add(this.phoneNumber +
32             "," + timeStart +
33             "," + timeEnd +
34             "," + this.callTo);
35     }
36 }
```

往ArrayList中添加记录, 注意对比StringBuffer

## 代码就像这个样子

```
56 //打印通话记录
57 void printDetails() {
58     //获取记录数目,即communicationRecords集合中的元素个数
59     int arrayListSize = this.communicationRecords.size();
60
61     for(int i = 0; i < arrayListSize - 1; i++) {
62         System.out.println("-----通话记录分割线-----");
63         String [] recordField = ((String) this.communicationRecords.get(i)).
64         System.out.println("主叫: " + recordField[0]);
65         System.out.println("被叫: " + recordField[3]);
66         Date timeStart = new Date(Long.parseLong(recordField[1]));
67         Date timeEnd = new Date(Long.parseLong(recordField[2]));
68         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月dd");
69
70         //SimpleDateFormat
71         System.out.println("通话开始时间: " + simpleDateFormat.format(timeStart));
72         System.out.println("通话结束时间: " + simpleDateFormat.format(timeEnd));
73
74         System.out.println("计费: "
75             + accountFee(Long.parseLong(recordField[1]), Long.parseLong(
76             + " 元。"));
77     }
78 }
79
```

注意代码改动的地方,右边没显示完整的应该怎么补充?

ArrayList实现了List接口，因此除了add(Object o)，它还有很多其他方法，p237-p238

■ **get(int index)**

获取，访问指定位置的元素

■ **remove(int index)**

删除

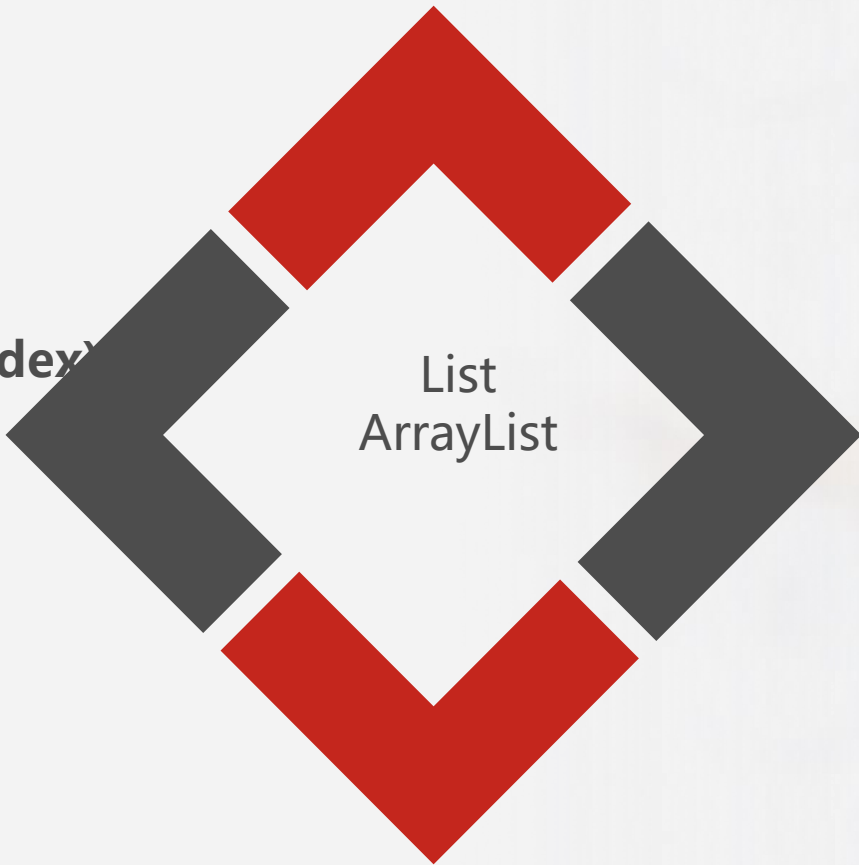
■ **clear()**

清空

■ **add(int index, Object o)**

在指定位置插入

List  
ArrayList





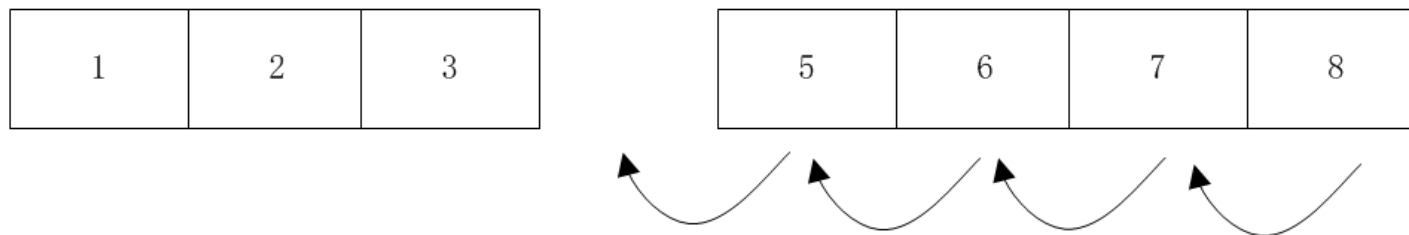
## 讨论：在ArrayList中添加、删除的效率

问题

删除前

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

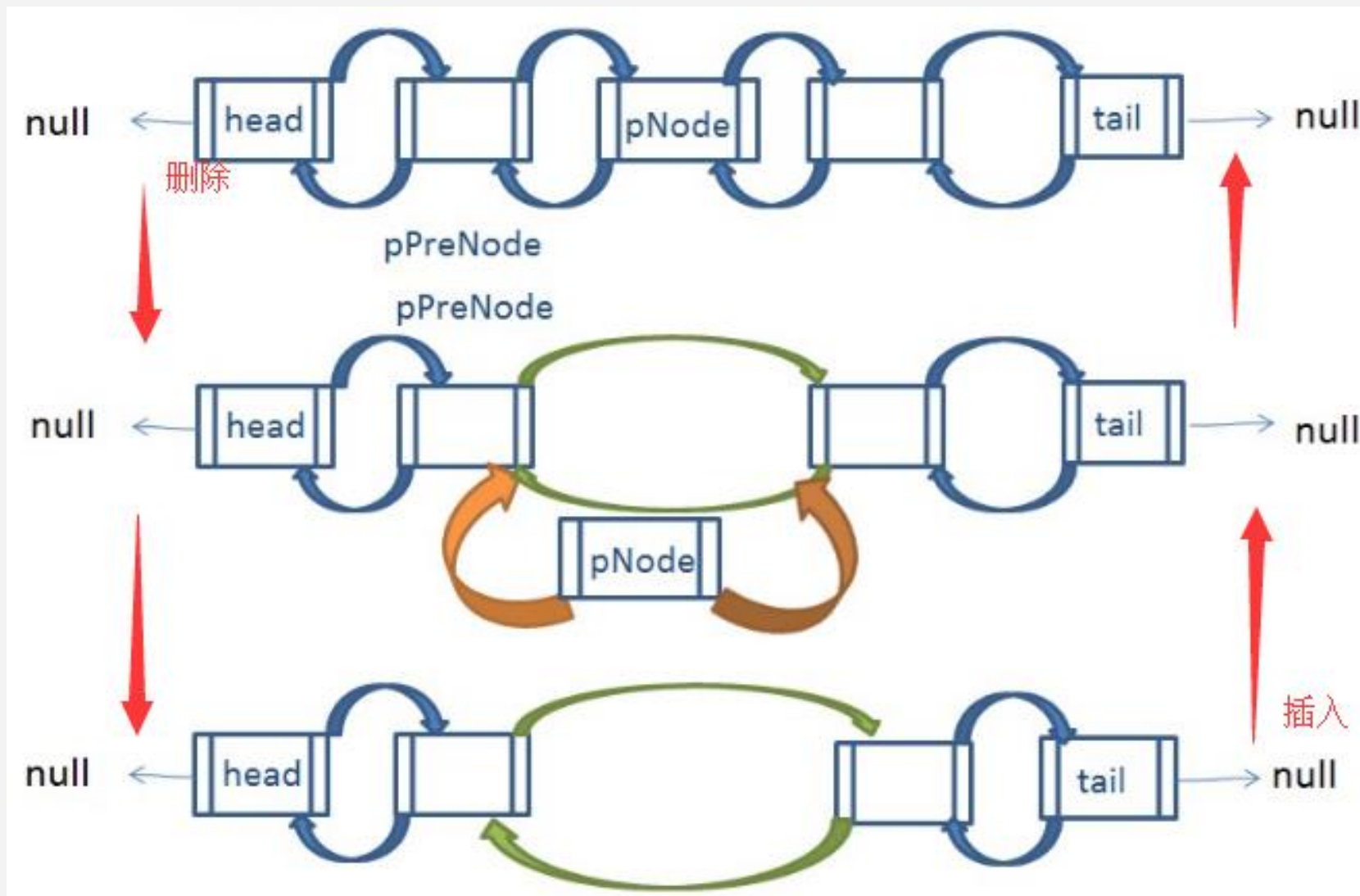
删除中



删除后

1	2	3	5	6	7	8
---	---	---	---	---	---	---

## 解决之道——LinkedList

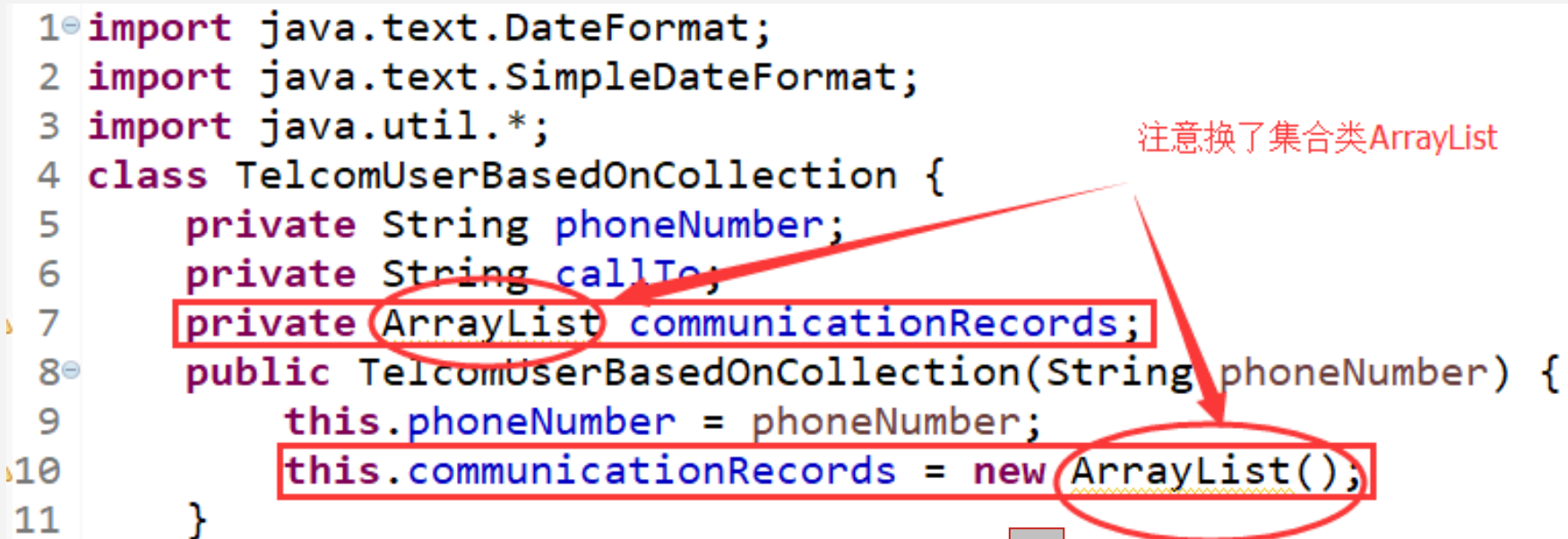




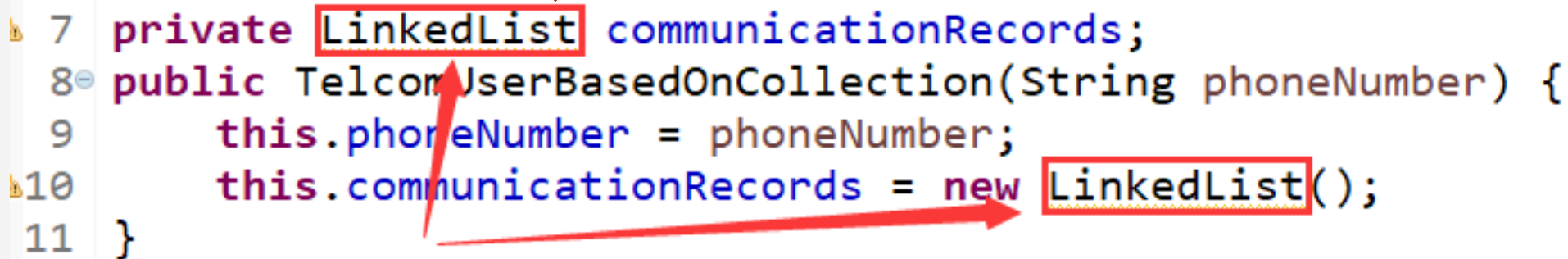
# 使用LinkedList模拟电信计费系统

```
1 import java.text.DateFormat;
2 import java.text.SimpleDateFormat;
3 import java.util.*;
4 class TelcomUserBasedOnCollection {
5     private String phoneNumber;
6     private String callTo;
7     private ArrayList communicationRecords;
8     public TelcomUserBasedOnCollection(String phoneNumber) {
9         this.phoneNumber = phoneNumber;
10        this.communicationRecords = new ArrayList();
11    }
```

注意换了集合类ArrayList



```
7 private LinkedList communicationRecords;
8 public TelcomUserBasedOnCollection(String phoneNumber) {
9     this.phoneNumber = phoneNumber;
10    this.communicationRecords = new LinkedList();
11 }
```



关于集合中元素的访问，可以当成数组，比如就像打印清单部分的程序这样：

```
56 //打印通话记录
57 void printDetails() {
58     //获取记录数目，即communicationRecords集合中的元素个数
59     int arrayListSize = this.communicationRecords.size();
60
61     for(int i = 0; i < arrayListSize - 1; i++) {
62         System.out.println("-----通话记录分割线-----");
63         String [] recordField = ((String) this.communicationRecords.get(i)).
64         System.out.println("主叫: " + recordField[0]);
65         System.out.println("被叫: " + recordField[3]);
66         Date timeStart = new Date(Long.parseLong(recordField[1]));
67         Date timeEnd = new Date(Long.parseLong(recordField[2]));
68         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月dd
69
70         //SimpleDateFormat
71         System.out.println("通话开始时间: " + simpleDateFormat.format(timeStart))
72         System.out.println("通话结束时间: " + simpleDateFormat.format(timeEnd));
73
74         System.out.println("计费: "
75             + accountFee(Long.parseLong(recordField[1]), Long.parseLong(
76             + " 元。"));
77     }
78 }
79
```

注意代码改动的地方，右边没显示完整的应该怎么补充？

或者，还可以使用“迭代器”，专门对付这种“遍历”需求

```
57 void printDetails() {
58     /*
59     * 使用迭代器
60     * */
61     Iterator it = this.communicationRecords.iterator();
62     while(it.hasNext()) {
63         System.out.println("-----通话记录分割线-----");
64         String [] recordField = ((String) it.next()).split(",");
65         System.out.println("主叫: " + recordField[0]);
66         System.out.println("被叫: " + recordField[3]);
67         Date timeStart = new Date(Long.parseLong(recordField[1]));
68         Date timeEnd = new Date(Long.parseLong(recordField[2]));
69         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月");
70
71         //SimpleDateFormat
72         System.out.println("通话开始时间: " + simpleDateFormat.format(timeStart));
73         System.out.println("通话结束时间: " + simpleDateFormat.format(timeEnd));
74
75         System.out.println("计费: "
76             + accountFee(Long.parseLong(recordField[1]), Long.parseLong(
77             + " 元。"));
78     }
79 }
```

如果用ListIterator, 还可以反向遍历, 请对比:

```
57 void printDetails() {
58     /*
59     * 使用ListIterator迭代器反向遍历
60     * */
61     ListIterator it = this.communicationRecords.listIterator(
62         this.communicationRecords.size());
63     while(it.hasPrevious()) {
64         System.out.println("-----通话记录分割线-----");
65         String [] recordField = ((String) it.previous()).split(",");
66         System.out.println("主叫: " + recordField[0]);
67         System.out.println("被叫: " + recordField[3]);
68         Date timeStart = new Date(Long.parseLong(recordField[1]));
69         Date timeEnd = new Date(Long.parseLong(recordField[2]));
70         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月");
71
72         //SimpleDateFormat
73         System.out.println("通话开始时间: " + simpleDateFormat.format(timeStart));
74         System.out.println("通话结束时间: " + simpleDateFormat.format(timeEnd));
75
76         System.out.println("计费: "
77             + accountFee(Long.parseLong(recordField[1]), Long.parseLong(recordField[2]))
78             + " 元。");
79     }
80 }
```

反向迭代的起始位置确定

## 嫌迭代器繁琐？ 可以用foreach循环

```
56 //打印通话记录
57 void printDetails() {
58     /*使用foreach循环遍历*/
59
60     for(Object aRecord:this.communicationRecords) {
61         System.out.println("-----通话记录分割线-----");
62         String [] recordField = ((String)aRecord).split(",");
63         System.out.println("主叫: " + recordField[0]);
64         System.out.println("被叫: " + recordField[3]);
65         Date timeStart = new Date(Long.parseLong(recordField[1]));
66         Date timeEnd = new Date(Long.parseLong(recordField[2]));
67         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");
68
69         //SimpleDateFormat
70         System.out.println("通话开始时间: " + simpleDateFormat.format(timeStart));
71         System.out.println("通话结束时间: " + simpleDateFormat.format(timeEnd));
72
73         System.out.println("计费: "
74             + accountFee(Long.parseLong(recordField[1]), Long.parseLong(recordField[2]))
75             + " 元。");
76     }
```

使用foreach遍历



或者用Vector(和ArrayList一样, 是List的一个实现类), 然后使用Enumeration接口

```
56 //打印通话记录
57 void printDetails() {
58     /*
59      * 使用Enumeration接口遍历。注意应将communicationRecords定义为Vector
60      */
61     Enumeration enumeration = this.communicationRecords.elements();
62     while(enumeration.hasMoreElements()) {
63         System.out.println("-----通话记录分割线-----");
64         String [] recordField = ((String)enumeration.nextElement()).split(" ");
65         System.out.println("主叫: " + recordField[0]);
66         System.out.println("被叫: " + recordField[3]);
67         Date timeStart = new Date(Long.parseLong(recordField[1]));
68         Date timeEnd = new Date(Long.parseLong(recordField[2]));
69         SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月c");
70
71         //SimpleDateFormat
72         System.out.println("通话开始时间: " + simpleDateFormat.format(timeStart));
73         System.out.println("通话结束时间: " + simpleDateFormat.format(timeEnd));
74
75         System.out.println("计费: "
76             + accountFee(Long.parseLong(recordField[1]), Long.parseLong
77             + " 元。");
```

新需求：单独收集通话记录中的被叫号码到一个集合，不允许重复。使用List行否？

```
7     private Vector communicationRecords;
8     private ArrayList callToNumbers;
9     public TelecomUserBasedOnCollection(String phoneNumber) {
10         this.phoneNumber = phoneNumber;
11         this.communicationRecords = new Vector();
12         this.callToNumbers = new ArrayList();
13     }
```

```
15     //模拟通话记录的生成
16     void generateCommunicateRecord() {
17         //随机生成通话记录数目
18         int recordNum = new Random().nextInt(10);
19         for(int i = 0; i <= recordNum; i++) {
20             //随机生成第i条通话记录
21             //用Calendar获取当前时间
22             Calendar cal = Calendar.getInstance();
23             //随机减去若干小时（10小时以内）
24             cal.add(Calendar.HOUR, - new Random().nextInt(10));
25             //获得对应毫秒
26             long timeStart = cal.getTimeInMillis();
27             //结束时间开始后的十分钟内随机的一个时间，至少一分钟
28             long timeEnd = timeStart + 60000 + new Random().nextInt(600000);
29             //被叫号码
30             this.callTo = getCallToPhoneNumber();
31             this.callToNumbers.add(this.callTo);
32             //插入通话记录
33             this.communicationRecords.add(this.phoneNumber +
34                 "," + timeStart +
35                 "," + timeEnd +
36                 "," + this.callTo);
37         }
38     }
```

13803720009  
13803720005  
13803720008  
13803720002  
13803720004  
13803720005  
13803720007  
13803720001  
13803720009  
13803720003

用Set接口可以去重，比如HashSet这个实现类。像下面这样

```
8     private HashSet callToNumbers;  
9     public TelecomUserBasedOnCollection(String phoneNumber) {  
10         this.phoneNumber = phoneNumber;  
11         this.communicationRecords = new Vector();  
12         this.callToNumbers = new HashSet();  
13     }
```

可否设计实验，做类似右边的List/Set验证呢？

13803720002  
13803720007  
13803720000  
13803720006  
13803720007  
13803720000  
13803720006

-----List/Set分割线-----

13803720002  
13803720000  
13803720007  
13803720006



如果还需要对这些号码进行排序呢？

```
8 private TreeSet callToNumbersSet;  
9 private ArrayList callToNumbersList;  
10 public TelcomUserBasedOnCollection(String phoneNumber) {  
11     this.phoneNumber = phoneNumber;  
12     this.communicationRecords = new Vector();  
13     this.callToNumbersSet = new TreeSet();  
14     this.callToNumbersList = new ArrayList();  
15 }
```

13803720004  
13803720000  
13803720004  
13803720008  
13803720003  
13803720002  
13803720004  
13803720002  
13803720009  
13803720009  
-----List/Set分割线-----  
13803720000  
13803720002  
13803720003  
13803720004  
13803720008  
13803720009

去重，有序。TreeSet



## 拓展思考

HashSet中的数据不会重复，如果输入的是字符串、整数还好理解，如果输入的是一个Java对象呢？怎么判断重复？

关注：P251-252

TreeSet中的数据会拥有平衡二叉树的排序特性，这样能够对其中的数据进行排序。字符串、整数好理解，如果输入的是一个Java对象呢？怎么排序，规则呢？

关注：P255-256

# 关于Java中的equals方法

## 所有Java的Object都有的方法

源码

```
public boolean equals(Object obj)
{
    return (this == obj);
}
```

## 默认情况下

两个对象必须是同一地址引用, equals的值才为真

举例class A {}

A a1 = new A();

A a2 = new A();

a1.equals(a2)结果: false

a1 == a2结果: false

## 特殊情况1

A是String

即String a1 = new String("hello")

String a2 = new String("hello")

A是包装类

现象

a1.equals(a2)结果: true

原因

String重写了默认的equals方法, 只比较内容

其他包装类类似

a1 == a2结果: false (原因是a1, a2是两个不同的对象, 内存地址不一样)

## 特殊情况2

String a1 = "hello";

String a2 = "hello"

现象

a1.equals(a2)结果: true

a1 == a2结果: true

原因

Java对常量的管理

相同常量来自内存相同位置 (所以a1 == a2结果为真, 因为内存地址一致)

```
A.java Test.java
6      //默认的一般情况
7      A a1 = new A();
8      A a2 = new A();
9      System.out.println("默认一般的对象比较: ");
10     System.out.println(a1.equals(a2));
11     System.out.println(a1 == a2);
12     //特殊情况1
13     String s1 = new String("hello");
14     String s2 = new String("hello");
15     System.out.println("特殊情况1——两个字符串对象的比较: ");
16     System.out.println(s1.equals(s2));
17     System.out.println(s1 == s2);
18     Integer aIntObj1 = new Integer(10);
19     Integer aIntObj2 = new Integer(10);
20     System.out.println("特殊情况1——两个整数对象的比较: ");
21     System.out.println(aIntObj1.equals(aIntObj2));
22     System.out.println(aIntObj1 == aIntObj2);
23     //特殊情况2
24     String s3 = "hello";
25     String s4 = "hello";
26     System.out.println("特殊情况2——两个字符串对象的比较(常量赋值): ");
27     System.out.println(s3.equals(s4));
28     System.out.println(s3 == s4);
29
```

```
Problems @ Javadoc Declaration Console
<terminated> Test [Java Application] C:\Program Files\Java\jre
默认一般的对象比较:
false
false
特殊情况1——两个字符串对象的比较:
true
false
特殊情况1——两个整数对象的比较:
true
false
特殊情况2——两个字符串对象的比较(常量赋值):
true
true
```

