Java容器

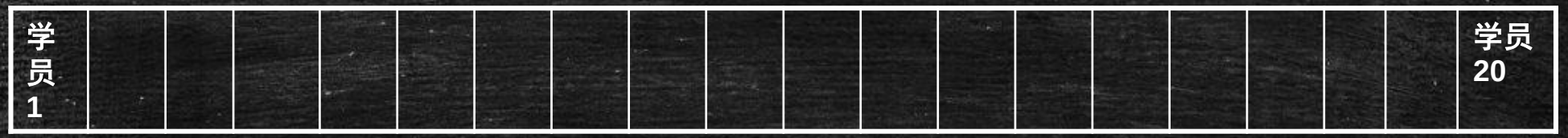
- What?Why?How?

本章概述

- 容器的概念
- 容器 API
- Collection 接口
- Iterator 接口
- Iterable接口
- Set 接口
- Comparable 接口
- List 接口
- Map 接口

为什么使用集合框架

- 存储一个班学员信息，假定一个班容纳20名学员

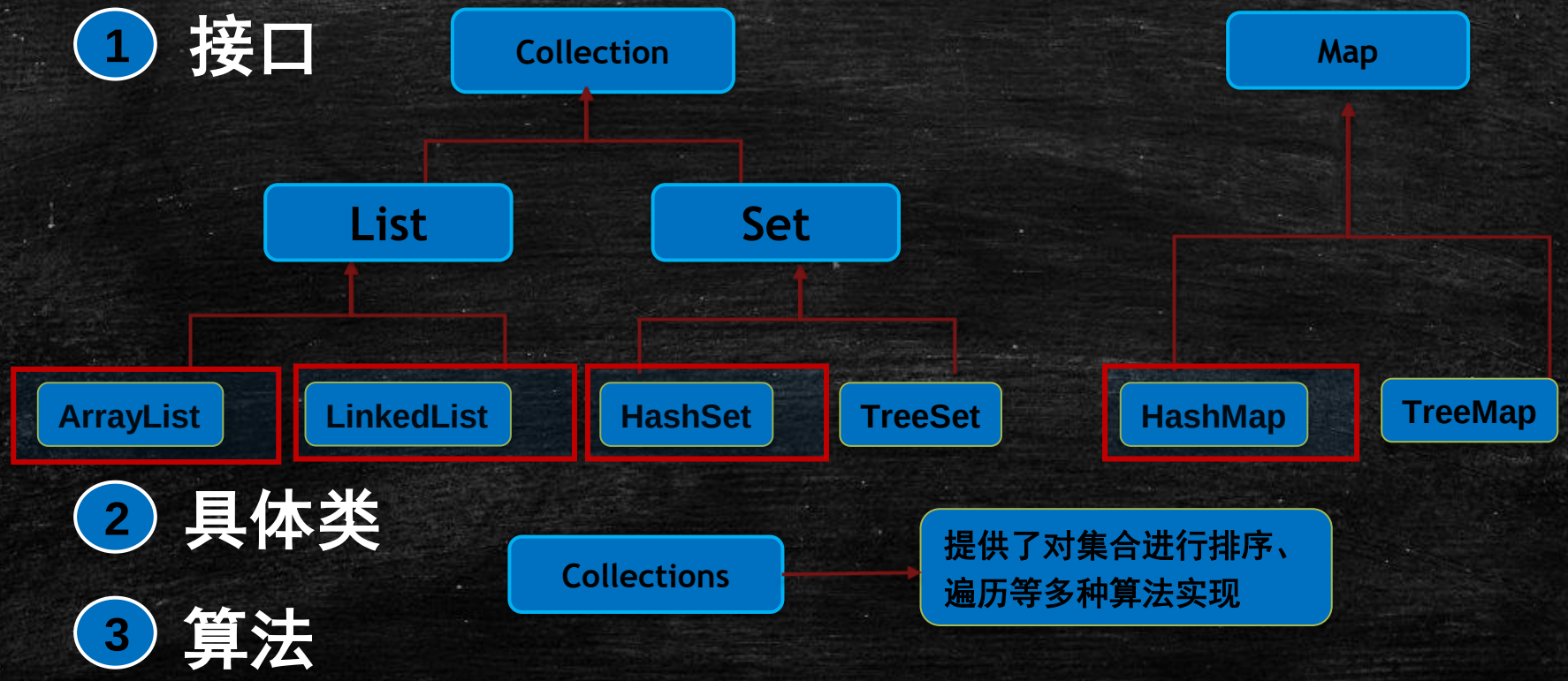


- 如何存储每天的新闻信息？

如果并不知道程序运行时会需要多少对象，或者需要更复杂方式存储对象——可以使用Java集合框架

Java集合框架包含的内容

- Java集合框架提供了一套性能优良、使用方便的接口和类，它们位于java.util包中



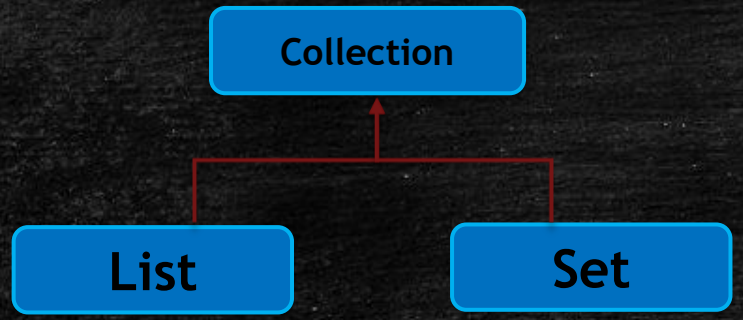
Collection接口的常用方法

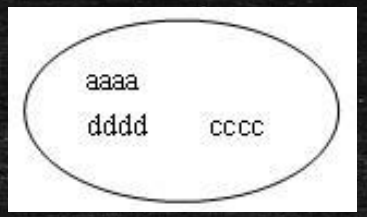
- 集合作为容器应该具有的功能（增，删，改，查），
- 不一定全有。
- 集合的基本操作：增加，删除，判断，取出

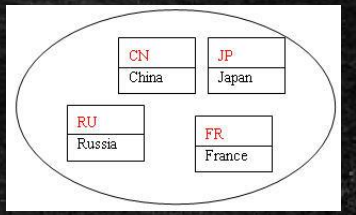
序号	方法名	作用
1	add(Object obj)	添加,存储的是对象的引用
2	size()	容器中元素的实际个数
3	remove(Object obj) clear() removeAll(Collection<?> c) retainAll(Collection<?> c)	删除
4	contains(Object obj) isEmpty()	判断元素
5	iterator()	遍历元素

List与Set接口

- Collection 接口存储一组不唯一，无序的对象
- List 接口存储一组不唯一，有序（插入顺序）的对象
- Set 接口存储一组唯一，无序的对象
- Map接口存储一组键值对象，提供key到value的映射



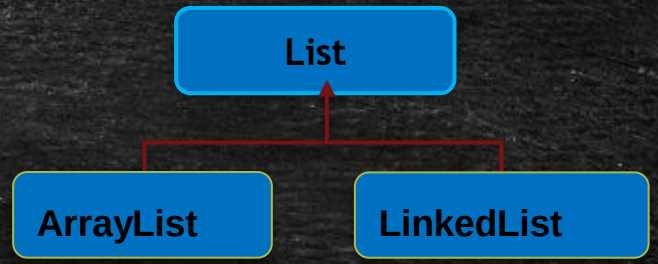




0	1	2	3	4	5	
aaaa	dddd	cccc	aaaa	eeee	dddd	

List接口的实现类

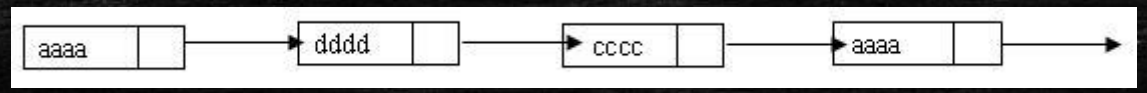
- List特点:有序，不唯一（可重复）



- ArrayList实现了长度可变的数组，在内存中分配连续的空间。
  - 优点：遍历元素和随机访问元素的效率比较高
  - 缺点：添加和删除需要大量移动元素效率低，按照内容查询效率低

0	1	2	3	4	5	
aaaa	dddd	cccc	aaaa	eeee	dddd	

- LinkedList采用链表存储方式。
  - 优点：插入、删除元素时效率比较高
  - 缺点：遍历和随机访问元素效率低下



List接口特有的方法

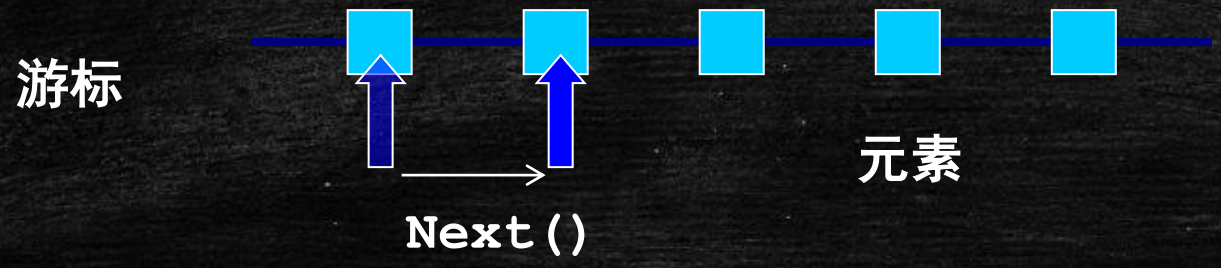
- 凡是可以操作索引的方法都是该体系特有方法

序号	方法名	作用
1（增）	add(index,element) addAll(index,Collection) addAll(Collection)	在指定索引的位置上插入元素 在指定的引的位置上插入整个集合的元素 在结束插入整个集合的元素
2（删）	remove(index)	根据索引删除指定的元素
3（改）	set(index,element)	使用element替换指定索引位置上的元素
4（查）	get(index) subList(from,to) listIterator();	获取元素

Iterator 接口

- 所有实现了Collection接口的容器类都有一个iterator方法用以返回一个实现了Iterator接口的对象。
- Iterator对象称作迭代器，用以方便的实现对容器内元素的遍历操作。
- Iterator接口定义了如下方法：

```
boolean hasNext();   //判断是否有元素没有被遍历
Object next();       //返回游标当前位置的元素并将游标移动到下一个位置
void remove();       //删除游标左面的元素，在执行完next之后该
                     //操作只能执行一次
```



Iterator接口

- 所有的集合类均未提供相应的遍历方法，而是把遍历交给迭代器完成。迭代器为集合而生，专门实现集合遍历
- Iterator是迭代器设计模式的具体实现
- Iterator方法
  - boolean hasNext():判断是否存在另一个可访问的元素
  - Object next():返回要访问的下一个元素
  - void remove():删除上次访问返回的对象
- 可以使用Iterator遍历的本质是什么?
  - 实现Iterable接口

Iterator

- For-each循环
  - 增强的for循环，遍历array或Collection的时候相当简便
  - 无需获得集合和数组的长度，无需使用索引访问元素，无需循环条件
  - 遍历集合时底层调用Iterator完成操作
- For-each缺陷
  - 数组：
    - 不能方便的访问下标值
    - 不要在for-each中尝试对变量赋值，只是一个临时变量
  - 集合：
    - 与使用Iterator相比，不能方便 的删除集合中的内容
- For-each总结
  - 除了简单的遍历并读出其中的内容外，不建议使用增强for

为什么需要ListIterator

- 在迭代过程中，准备添加或者删除元素

```
ArrayList al=new ArrayList();
al.add("java1");//添加元素
al.add("java2");
al.add("java3");
//遍历
Iterator it=al.iterator();
while(it.hasNext()){
      Object obj=it.next();
      if (obj.equals("java2")) {
                  al.add("java9");
      }
     sop("obj="+obj);
}
```

ListIterator的作用→解决并发操作异常

- 在迭代时，不可能通过集合对象的方法(al.add(?))操作集合中的元素，
- 会发生并发修改异常。
- 所以，在迭代时只能通过迭代器的方法操作元素，但是Iterator的方法
- 是有限的，只能进行判断(hasNext)，取出(next),删除(remove)的操作，
- 如果想要在迭代的过程中进行向集合中添加，修改元素等就需要使用
- ListIterator接口中的方法

```
ListIterator li=al.listIterator();
while(li.hasNext()){
    Object obj=li.next();
    if ("java2".equals(obj)) {
        li.add("java9994");
        li.set("java002");
     }
}
```

上机练习一

- 创建一个Dog类，包含昵称，亲密度两上属性，
- 创建测试类，完成以下任务
- 需求说明：
  - 把多个Dog的信息添加到集合中
  - 查看Dog的数量及所有Dog的信息
  - 删除集合中部分Dog的元素
  - 判断集合中是否包含指定Dog

```
共有:4只狗狗
分别是:
昵称        亲密度
欧欧        90
美美        100
丽丽        89
亚亚        99
==============================
删除之后，还有:2只狗狗
分别是:
昵称        亲密度
欧欧        90
亚亚        99
==============================
集合中包含亚亚的信息
```

List接口的实现类LinkedList

- linkedList特有的方法

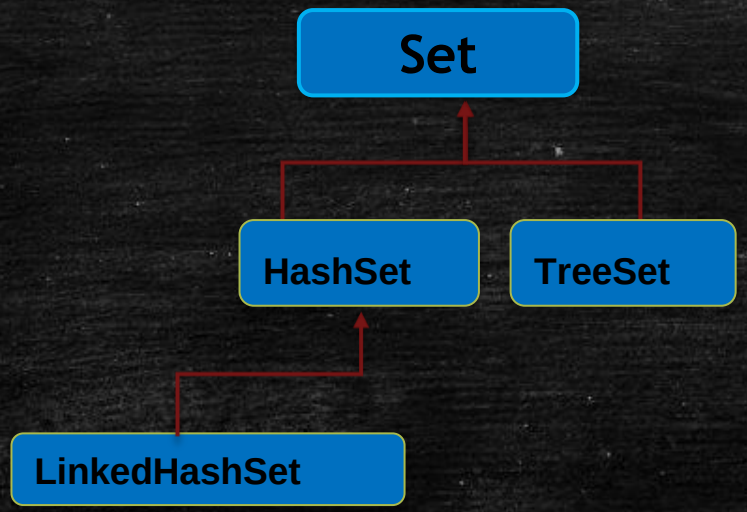
序号	方法名	作用
1（增）	addFirst(Object obj) addLast(Object obj) offerFirst(Object obj) offerLast(Object obj)	添加头 添加尾 1.6版本之后的加头，尾巴
2（删）	removeFirst() removeLast(); pollFirst() pollLast()	删除头　　获取元素并删除元素 删除尾 1.6版本之后的删头，删尾
3（查）	getFirst() getLast() peekFirst() peekLast()	获取头　　获取元素但不删除 获取尾 1.6版本之后的获取头，获取尾

小结

- **ArrayList**
  - 遍历元素和随机访问元素的效率比较高
  - 插入、删除等操作频繁时性能低下
- **LinkedList**
  - 插入、删除元素时效率较高
  - 查找效率较低

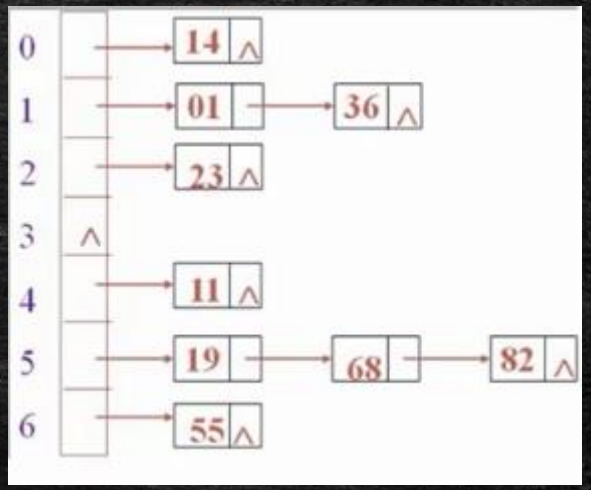
Set接口中的实现类

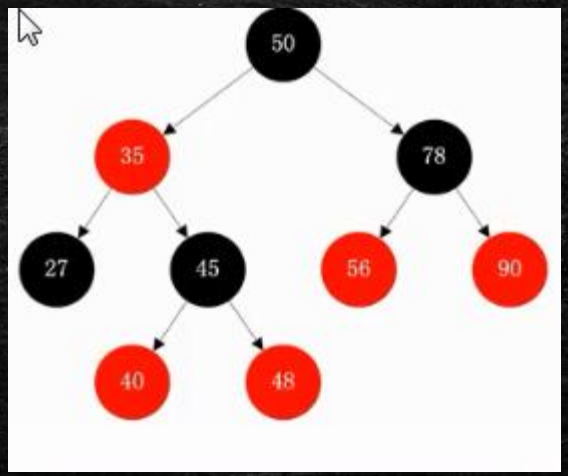
- Set接口
  - Set接口存储一组唯一，无序的对象
  - （存入和取出的顺序不一定一致）
  - 操作数据的方法与List类似，Set接口不存在get()方法



Set接口中的实现类

- HashSet:采用Hashtable哈希表存储结构
  - 优点：添加速度快，查询速度快，删除速度快
  - 缺点：无序
  - LinkedHashSet
    - 采用哈希表存储结构，同时使用链表维护次序
    - 有序（添加顺序）
- TreeSet
  - 采用二叉树（红黑树）的存储结构
  - 优点：有序（排序后的升序）查询速度比List快
  - 缺点：查询速度没有HashSet快



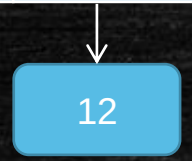


Hash表原理

num	45	33	12	45	32	56	90

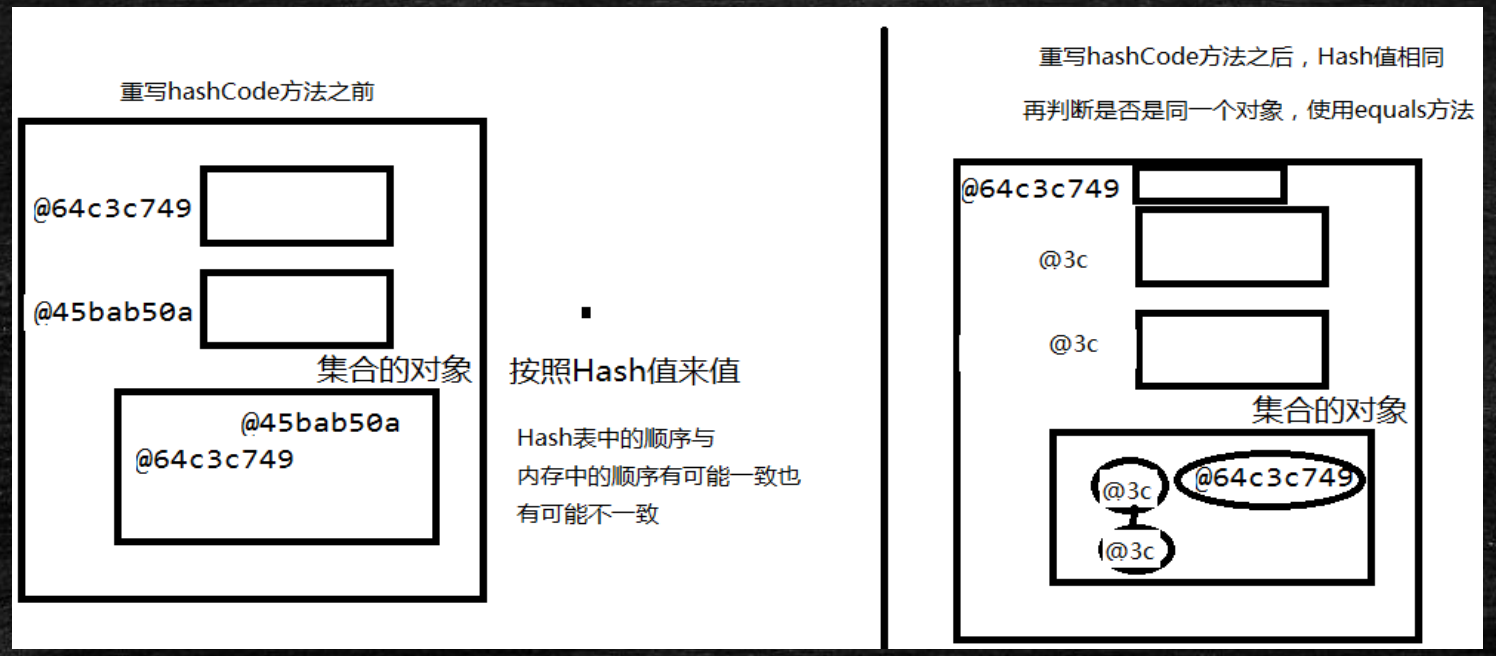
num	45	33	12	45	32	56	90
hashCode	45	33	12	45	32	56	90
Y=k(x)=x%7	3	5	5	3	4	0	6

0	1	2	3	4	5	6
56			45	32	33	90



Hash表

- 代码验证HashSet的无序性与唯一性
- 使用HashSet存储自定义对象，重写hashCode方法与equals方法



HashSet

- 关键代码

```
HashSet hs=new HashSet();//创建HashSet对象
hs.add(new Person("张三",20));
hs.add(new Person("李四",22));
hs.add(new Person("王五",23));
hs.add(new Person("李四",22));
```

```
李四        22
张三        20
李四        22
王五        23
```

- HashSet存储进了相同的对象，不符合实际情况
- **解决方案**:
- 重写equals方法与hashCode方法

HashSet的操作

```
@Override
public int hashCode() {
    System.out.println(this.name+".....hashCode");
    return 60;
}
```

```
张三.....hashCode
李四.....hashCode
李四...equals张三
王五.....hashCode
王五...equals李四
王五...equals张三
李四.....hashCode
李四...equals王五
李四...equals李四
王五      23
李四      22
张三      20
```

HashSet操作

- hashCode都相同，不符合实际情况，继续升级
- 修改hashCode方法

```
@Override
public int hashCode() {
    System.out.println(this.name+".....hashCode");
     return this.name.hashCode()+age;
}
```

```
张三.....hashCode
李四.....hashCode
王五.....hashCode
李四.....hashCode
李四...equals李四
张三        20
李四        22
王五        23
```

HashSet操作

- **总结:**
- HashSet是如何保证元素的唯一性的呢?
- 答:是通过元素的两个方法,hashCode和equals方法来完成
- 如果元素的HashCode值相同，才会判断equals是否为true
- 如果元素的hashCode值不同，不会调用equals方法

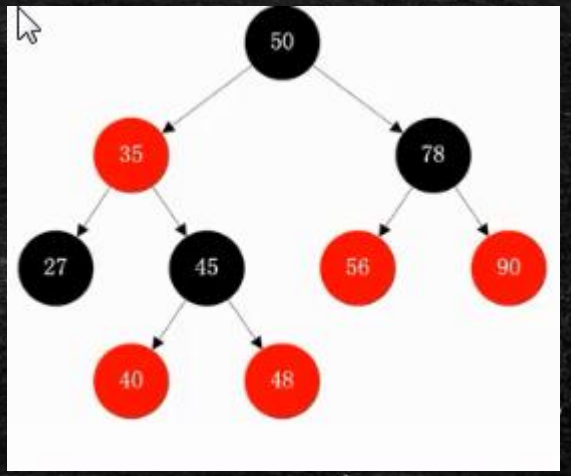
上机练习三

- 需求分析：
- 创建员工类，包含如下信息(工号，姓名，年龄)
- 使用HashSet集合存储，要求保证员工对象的唯一性
- 【1】输出员工的信息
- 【2】删除离职的员工
- 【3】判断指定员工是否存在

```
本单位共有员工人数为:3
人员内容分别如下:
1002      李默文     22
1001      王小华     20
1003      张一慢     24
=========================================
本单位共有员工人数为:2
人员内容分别如下:
1002      李默文     22
1003      张一慢     24
=========================================
判断张一慢是否存在
存在该员工
```

TreeSet

- TreeSet
  - 采用二叉树（红黑树）的存储结构
  - 优点：有序（排序后的升序）查询速度比List快
  - 缺点：查询速度没有HashSet快



Comparable 接口

- 问题：上面的算法根据什么确定集合中对象的“大小”顺序？
- 所有可以“排序”的类都实现了java.lang.Comparable 接口，Comparable接口中只有一个方法

  public int compareTo(Object obj);

  该方法:

  - 返回 0 表示 this == obj
  - 返回正数 表示 this > obj
  - 返回负数 表示 this < obj

- 实现了Comparable 接口的类通过实现 comparaTo 方法从而确定该类对象的排序方式。

sort排序

```
public class StrLenComparator implements Comparator<String> {
        @Override
        public int compare(String o1, String o2) {
                if (o1.length()>o2.length()) {
                        return 1;
                }
                if (o1.length()<o2.length()) {
                        return -1;
                }
                return o1.compareTo(o2);//长度相同，按字母
        }
}
```

```
public static void sortDemo(){
        List<String> list=new ArrayList<String>();
        ..添加元素
        sop(list);
        Collections.sort(list);//按字母排序
        sop(list);
        //按照字符串长度排序
        Collections.sort(list,new StrLenComparator());
        sop(list);
}
```

泛型

- 为什么需要泛型
- 解决数据类型操作不统一产生的异常
- 使用泛型可以更好的去保护数据类型
- 泛型类的定义

```
.....
package cn.mashibing.demo;
public class Point2<T> {//此处可以随便写标识符号,T是type的简称,也可以写a,b,c
        private T var;//var的类型由T指写,由外部指定
        public T getVar() {//返回值的类型由外部指定
                return var;
        }
        public void setVar(T var) {//参数的类型由外部指定
                this.var = var;
        }
} .....
```

泛型类的定义

```
.....
public class Notepad<K,V> {// 此处指定了两个泛型
        private K key;//此变量的类型由外部决定
        private V value;//此变量的类型由外部决定
.....
}
```

```
.....
Notepad<String,Integer>t=null;//指定两个泛型类型的对象
     //Key为String,Value为Integer
      t=new Notepad<String,Integer>();
      t.setKey("张三");
      t.setValue(30);
System.out.println("姓名:"+t.getKey()+"\t年龄:"+t.getValue());
```

使用泛型集合解决实际问题

- 声明员工类Employee包含如下属性:id,name,age,gender(枚举类型)
- 声明程序员类SE,含有属性popularity人气值
- 声明项目经理类PM,含有属性workOfYear工作年限
- 程序员与项目前经理都继承自Employee
- **需求说明:**
- 使用泛型集合ArrayList,LinkedList,HashSet,TreeSet完成员工的添加，删除，
- 判断，集合中元素个数的判断