第一章：绪论

1、计算机科技的两大支柱 1、数据结构 2、算法

· 程序 ＝ 数据结构 ＋ 算法

2、**数据结构定义**: 是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作等等的学科。

数据(Data):是对信息的一种符号表示。在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。

数据元素(Data Element):是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。

一个数据元素可由若干个数据项组成。数据项是数据的不可分割的最小单位。

数据结构(Data Structure): 是相互之间存在一种或多种特定关系的数据元素的集合。

3、数据结构主要指逻辑结构和物理结构

数据之间的相互关系称为逻辑结构。通常分为四类基本结构：

集合 结构中的数据元素除了同属于一种类型外，别无其它关系。

线性结构 结构中的数据元素之间存在一对一的关系。

树型结构 结构中的数据元素之间存在一对多的关系。

图状结构或网状结构 结构中的数据元素之间存在多对多的关系。

4、数据结构在计算机中有两种不同的表示方法：

顺序存储结构:用数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系。

链式存储结构：在每一个数据元素中增加一个存放地址的指针，用此指针来表示数据元素之间的逻辑关系。

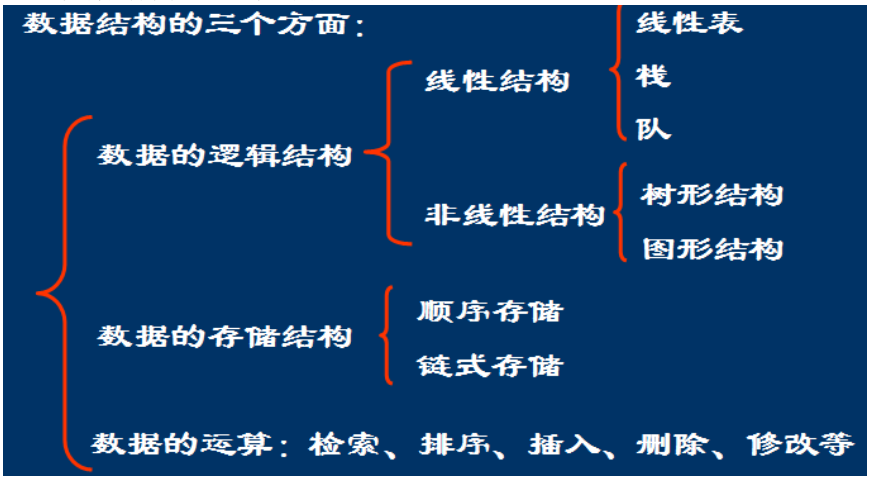
5、**数据对象**：某种数据类型元素的集合。

例：整数的数据对象是{…-3，-2，-1，0，1，2，3，…}

英文字符类型的数据对象是{A，B，C，D，E，F，…}

数据类型：在一种程序设计语言中，变量所具有的数据种类。

6、数据结构的三个方面：



7、**算法**用抽象的语言描述解决特定问题的每一步的操作。**程序**是计算机能理解和执行的指令序列。一个程序实现一个算法。算法和程序的区别是算法的执行是有穷的，而程序的执行可以是无限的。

8、时间复杂度：

时间复杂度为O(n^2)的二重循环。

```
int n=9;
for(int i=0;i<n;j++)
   for(int j=0;j<n;j++)
      System.out.print(j*j);
```

如果

```
int n=9;
for(int i=0;i<n;j++)
   for(int j=0;j<i;j++)
      System.out.print(j*j);
```

二重循环的执行次数为 ，时间复杂度仍为O(n^2)。

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

4. 时间复杂度为O($n\log_2 n$)的二重循环。

```
int n=8;
for(int i=1;i<=n;j*=2)
    for(int j=1;j<=n;j++)
        System.out.print(j*j);
```

循环次数为 $\sum_{i=1}^{\log_2 n} n = O(n \log_2 n)$ 。时间复杂度为O($n\log_2 n$)。

5. 时间复杂度为O(n)的二重循环。

```
int n=8;
for(int i=1;i<=n;j*=2)
    for(int j=1;j<=i;j++)
        System.out.print(j*j);
```

总的循环次数为 $\sum_{i=1}^{\log_2 n} 2^i = O(n)$ 。时间复杂度为O(n)。

9、**1、什么是集合**

通常情况下，把具有相同性质的一类东西，汇聚成一个整体，就可以称为集合。比如，用 Java 编程的所有程序员，全体中国人等。

2、什么是集合框架

集合框架是为表示和操作集合而规定的一种统一的标准的体系结构。任何集合框架都包含三大块内容：对外的接口、接口的实现和对集合运算的算法。

3、集合框架对我们编程有何助益：它减少了程序设计的辛劳、它提高了程序速度和质量。

10、Collection 接口是一组允许重复的对象。

Set 接口继承 Collection，但不允许重复，使用自己内部的一个排列机制。

List 接口继承 Collection，允许重复，以元素安插的次序来放置元素，不会重新排列。

Map 接口是一组成对的键一值对象，即所持有的是 key-value pairs。Map 中不能有重复的 key。拥有自己的内部排列机制。

容器中的元素类型都为 Object。从容器取得元素时，必须把它转换成原来的类型。

第四章：递归

1、递归的定义　若一个对象部分地包含它自己，或用它自己给自己定义，则称这个对象是递归的；若一个过程直接地或间接地调用自己，则称这个过程是递归的过程。

第五章：数组列表

1、线性表是由*n*（*n*≥0）个相同类型的数据元素*a*1，*a*2，…，*a*n组成的有限序列，记作：LinearList={*a*1，*a*2，…，*a*n}

其中，*n* 表示线性表的元素个数，称为线性表的长度。

2、**线性表的顺序存储结构：**是用一组连续的存储单元顺序存放线性表的数据元素，数据元素在内存的物理存储次序与它们在线性表中的逻辑次序是一致的，即数据元素 *a*i 与其前驱数据元素 *a*i-1 及后继数据元素 *a*i+1 的位置相邻。

5、**为什么需要串行化：**希望 ArrayList 对象保存到文件中，以便于恢复使用。

6、**如何实现类可串行化：见练习**

```
package java.io;
public interface Serializable {//标记接口
}
```

7、迭代器是允许以一致的方式对集合对象的元素进行访问的对象。

迭代器对象一旦发现另一个对象在结构上修改这一集合，就马上会报错。这是因为一旦你开始对一个 ArrayList 对象进行迭代，就不能再修改这个 ArrayList 完整性。因此弹出 ConcurrentModificationException

1. 掌握 Collection 接口含义及方法

Collection接口

*Collection 是层次结构*中的根接口。Collection 表示一组对象，这些对象也称为 collection 的*元素*。JDK 不提供此接口的任何*直接*实现：它提供具体的子接口（如 Set 和 List）实现。

Collection方法：

```
package java.util;
public interface Collection {
  int size();//返回此 collection 中的元素数。
  boolean isEmpty();//如果此 collection 不包含元素，则返回 true。
  boolean contains(Object o);//如果此 collection 包含指定的元素，则返回 true。
  Iterator iterator();//返回在此 collection 的元素上进行迭代的迭代器。
  Object[] toArray();//返回包含此 collection 中所有元素的数组。
  Object[] toArray(Object a[]);// 返回包含此 collection 中所有元素的数组。
  boolean add(Object o);//确保此 collection 包含指定的元素（可选操作）。
  boolean remove(Object o);//从此 collection 中移除指定元素的单个实例，如果存在的话（可选操作）。
  boolean containsAll(Collection c);//如果此 collection 包含指定 collection 中的所有元素，则返回 true。
  boolean addAll(Collection c);//将指定 collection 中的所有元素都添加到此 collection 中（可选操作）。
  boolean removeAll(Collection c);//移除此 collection 中那些也包含在指定 collection 中的所有元素（可选操作）。
  boolean retainAll(Collection c);//仅保留此 collection 中那些也包含在指定 collection 的元素（可选操作）。
  void clear();//移除此 collection 中的所有元素（可选操作）。
  boolean equals(Object o);//比较此 collection 与指定对象是否相等。
  int hashCode();//返回此 collection 的哈希码值。
```

2. 掌握 Collection 接口与 List 接口的关系:见练习本

3. 掌握 ArrayList 与数组之间的关系：ArrayList 对数组进行了排序，见练习本
4. 从用户角度如何理解 ArrayList？从开发者角度如何理解 ArrayList？ArrayList 比数组多了一些增减数据的方法。
5. 掌握 ArrayList 常用方法的含义及使用方法。见书 P121
6. 会实现 ArrayList 的顺序存储简化版，包括最常用的方法 size()；isEmpty()；contains(Object o)；indexOf(Object elem)；add(Object o)；remove(Object o)；clear()；get (int index)；set (int index, Object element)等

（1） public int size(){ return size; }

（2） public boolean isEmpty(){
```
        if(size==0)
            return true;
        else
            return false;
    }
```

（3） public Boolean contains(Object o){
```
return indexOf(o)>=0;
}
```

（4）public int indexOf(Object elem){
```
    if (elem == null) {
        for (int i = 0; i < size; i++)
            if (elementDate[i]==null)
                return i;
    } else {
        for (int i = 0; i < size; i++)
            if (elem.equals(elementDate[i]))
                return i;
    }
    return -1;
}
```

（5）public boolean add(Object o){
```
    ensureCapacity(size+1);
    elementDate[size++]=o;
    return true;
}
```

（6）public Object remove(int index){
```
    if(index<0||index>=size)
        return null;
    Object temp=elementDate[index];
    for(int j=index;j<size;j++){
        elementDate[j]=elementDate[j+1];
    }
    size--;
    return temp;
}
```

（7）public void clear(){
```
    modCount++;
    for(int i=0;i<size;i++)
        elementData[i]=null;
    size=0;
    }
```

（8）public Object get(int index){
```
    if(index<0||index>=size)
        return null;
    return elementDate[index];

}
```

（9）

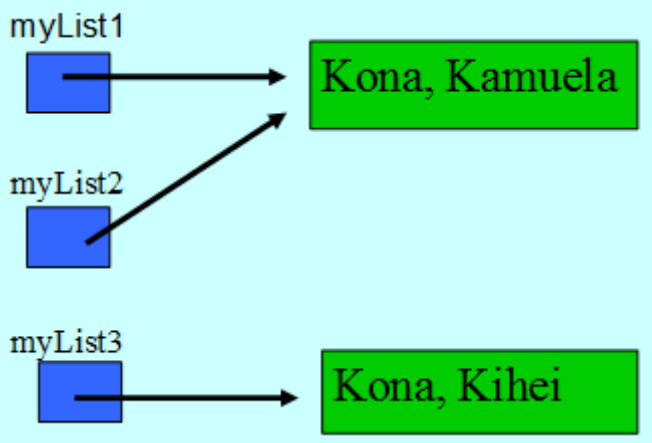
7. 理解 clone 方法的含义，并对浅克隆与深度克隆进行说明及图示

为什么需要克隆：希望获得一个副本，即另一份拷贝。

如何实现类可克隆

```
package java.lang;
public interface Cloneable { //标记接口
}
```

Eg:
```
public Object clone() {
    try {
        ArrayList v = (ArrayList)super.clone(); // copies size
        v.elementData = new Object[size];
        System.arraycopy(elementData, 0, v.elementData,0, size);
        v.modCount = 0;
        return v;
    }
    catch (CloneNotSupportedException e) {
        // this shouldn't happen, since we are Cloneable
        throw new InternalError();
    }
}
```



二、 常见题目

1. 教材上的习题、上机题、常用几个方法的时间复杂度

4. 阅读程序，写出运行结果

```
ArrayList let=new ArrayList();
let.add("p");
let.add(0, "r");
let.add("e");
System.out.println(let.indexOf("p"));
let.remove(1);
System.out.println(let.get(1));
System.out.println(let.contains("r"));
```

5. ArrayList 中定义了一个方法，说明该方法在类中正确的方法名称

```
public int unkown(Object elem) {
    if (elem == null) {
        for (int i = 0; i < size; i++)
        if (elementData[i]==null)
            return i;
    } else {
```

```
        for (int i = 0; i < size; i++)
        if (elem.equals(elementData[i]))
            return i;
    }
    return -1;
    }
```

6．假设向 ArrayList 中添加如下方法，写出每个方法的方法体

public boolean addFirst (Object element)
public boolean addLast (Object element)
public Object removeFirst()
public Object removeLast()
public Object getFirst ( )
public Object getLast ( )

第六章：链表

一、要求

1. 掌握对象变量与对象实体关系，图示说明简单变量与对象变量的区别
2. 以单链表为存储结构，实现 LinkedList 的简化版，包括最常用的方法 size()；isEmpty()；contains(Object o) ； indexOf(Object elem) ；add(Object o) ； remove(Object o) ； clear()；get (int index) ； set (int index, Object element)； get (int index); getFirst(),getLast()，removeFirst(),removeLast()等

```
public class MyLinkedList {
        Entry header;
        int size;
        public MyLinkedList (){
                header=new Entry(null,null);
                size=0;
        }
        public int size(){
                return size;
        }
        public boolean isEmpty(){
                if(size==0)
                        return true;
                else
                        return false;
        }
        public boolean add(Object o){
                Entry q=new Entry(o,null);
                Entry p=header;
                while(p.next!=null)
                        p=p.next ;
                p.next=q;
                size++;
                return true;
                }
                public void add(int index,Object element){
                        /*if(index<0||index>size)
                                throw IndexOutOfBoundsException(+size);*/
                        Entry q=new Entry(element,null);
                        Entry p=header;
                        for(int i=0;i<index;i++)
                                p=p.next;
```

```
                q.next=p.next;
                p.next=q;
                size++;
        }
public boolean contains(Object elem){
                Entry p=header.next;
                while(p!=null){
                        if(p.element.equals(elem))
                                return true;
                        p=p.next;
                }
                return false;
        }
/*public Object get(int index){
        if(index<0||index>=size)
                Entry q=header;
        for(int i=0;i<index;i++)
                q=q.next;
        return q;


}
*/
public Object remove(int index){
        if(index<0||index>=size)
                return null;
        Entry q=header;
        for(int j=index;j<size;j++){
                q=q.next;
                q.next=q.next.next;
        }
        size--;
        return q;
}
/*public int indexOf(Object elem){
        if (elem == null) {
              for (int i = 0; i < size; i++)
                    if (elementDate[i]==null)
                          return i;
        } else {
              for (int i = 0; i < size; i++)
                    if (elem.equals(elementDate[i]))
                          return i;
        }
        return -1;
```

3. 掌握单环链表定义并以单环链表的形式实现上题：见练习
4. 掌握双端链表定义并以双端链表的形式实现上题：见练习
5. 掌握双向链表、双向循环链表概念及定义；会画出双向循环链表的插入、删除图示以及会书写部分代码：见下
6. 从用户角度如何理解 LinkedList？从开发者角度如何理解 LinkedList？
7. 掌握 LinkedList 常用方法的含义及使用方法：见书 P150
8. 如何理解链式存储结构中设立哨兵（头结点）的方式

9. 如何理解 LinkedList 中 header 数据成员

10.

哨兵问题：

```
private transient Entry header = new Entry(null, null);
private transient int size = 0;
```

11. 简述 LinkedList 与 ArrayList 差别

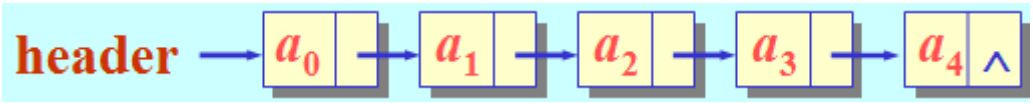
如果应用程序对各个索引位置的元素进行大量存取/删除操作，ArrayList 对象要远远优于 Linked List 对象.

如果应用程序主要对列表进行循环，并且在循环的时候进行插入或者删除操作，LinkedList 对象要远远优于 ArrayList 对象.

12. 你是如何理解迭代器工作？掌握迭代器的使用方法：见下

1. **单链表的特点：**每个元素(表项)由结点(*Node*)构成；

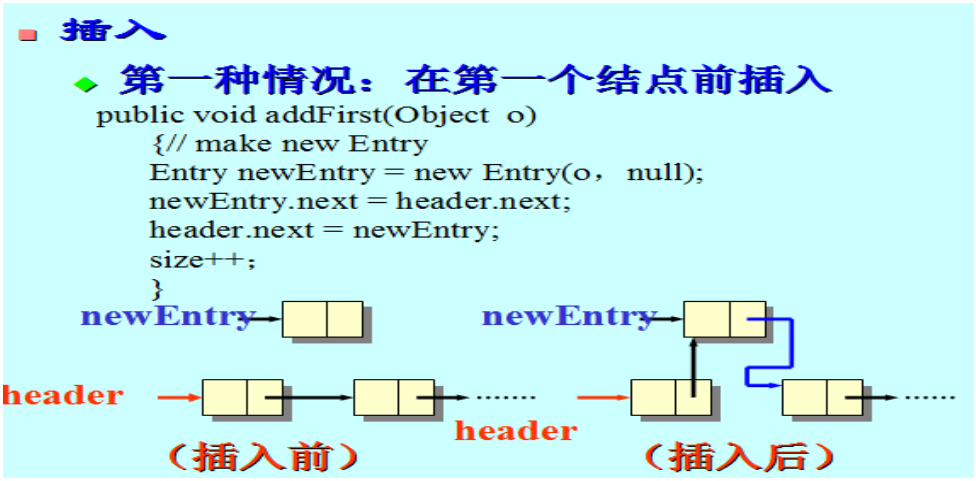
线性结构

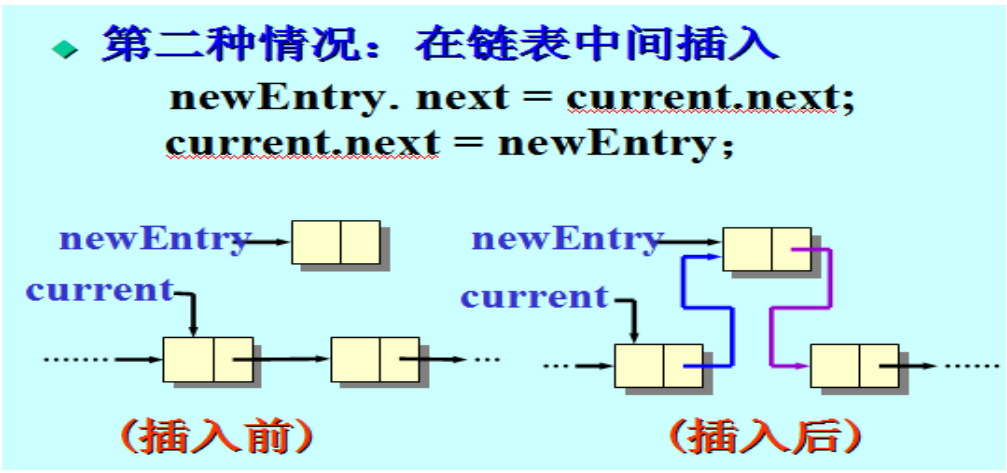


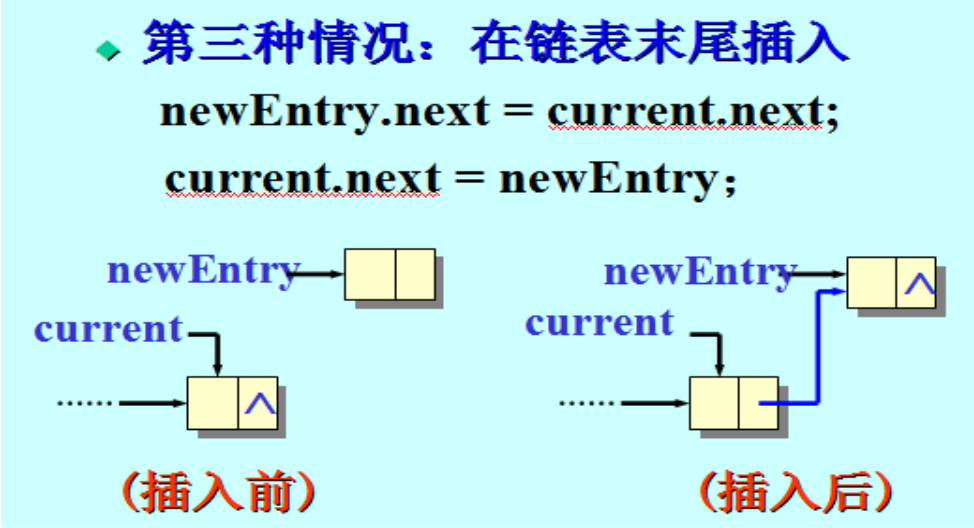
结点可以不连续存储

表可扩充。

2. 插入结点







3. 删除结点

```
(1) public Object removeFirst()
        {// (assumes list not empty)
        Object first= header.next.element;
        header.next = header.next.next;
         size--;
        return first;
        }
(2)public Object removeLast() {
            Entry    pre = header;
            Entry current=pre.next;
            while(current.next != null) {
                    pre= current;
                    current = current.next;
              }
               pre.next=null;
               size--;
  return current.element;
    }
```

4、遍历

为了显示链表，从 header 开始，沿着引用链从一个链结点到下一个链结点。变量 current 按顺序指向每一个链结点。Current 首先指向 header,那里有第一个链节点的引用。

```
  public void displayList()
    {
        System.out.print("List (header-->last): ");
        Entry current = header.next;            // start at beginning of list
        while(current != null)          // until end of list,
       {
            current.displayEntry();     // print data
            current = current.next;    // move to next Entry
       }
```

```
            System.out.println("");
```

5、查找

```
public int indexOf(Object o) {
            int index = 0;
            for (Entry e = header.next; e != null; e = e.next) {
                        if (o.equals(e.element))
                              return index;
                        index++;
            }
            return -1;
      }
public boolean contains(Object o) {
            return indexOf(o) != -1;
      }
```

6、获取

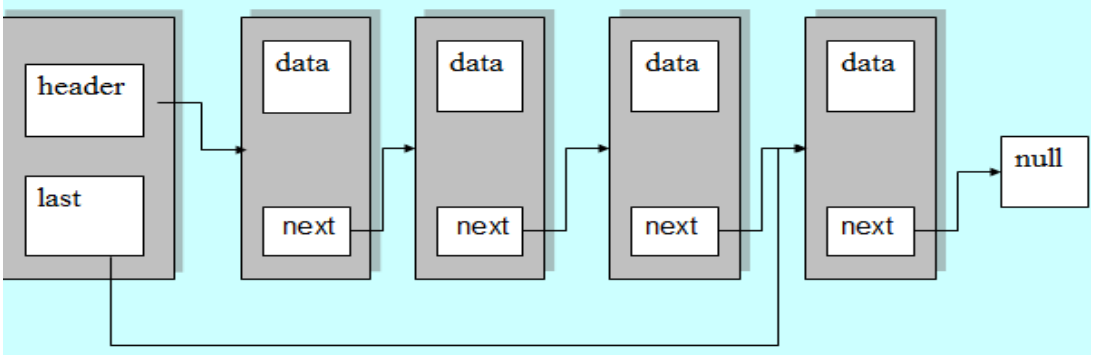
```
（1）public Object getFirst() {
  if (size= =0)
        throw new NoSuchElementException();

  return header.next.element;
      }
（2）public Object get(int index) {
            if (index < 0 || index >= size)
                  throw new IndexOutOfBoundsException("Index: "+index+
                                                      ", Size: "+size);
            Entry e = header.next;
            for (int i = 0; i < index; i++)
                        e = e.next;

            return       e.element;
      }
（3）public Object getLast() {
    ……..
}
```

7、双端链表

双端链表与传统的链表非常相似，但是它有一个新特性：对最后一个链结点的引用，就像对第一个链结点的引用一样。



8、迭代器

Iterator 接口

```
package java.util;
public interface Iterator {
boolean hasNext();
Object next();
void remove();
}
```

ListIterator 接口

```
package java.util;
public interface ListIterator extends Iterator {
    boolean hasNext();
    Object next();
    void remove();
    boolean hasPrevious();
    Object previous();
    int nextIndex();
    int previousIndex();
    void set(Object o);
    void add(Object o);
}
```

二、常见题目

1. 教材上的习题
2. 上机题
3. 常用几个方法的时间复杂度

4.阅读程序，写出运行结果

```
LinkedList let=new LinkedList();
let.add(“p”);
let.add(0, “r”);
let.add(“e”);
System.out.println(let.indexOf(“p”));
let.remove(1);
System.out.println(let.get(1));
System.out.println(let.contains(“r”));
System.out.println(let.getFirst());
```

5．LinkedList 中定义了一个方法，说明该方法在类中正确的方法名称

```
public int unkown(Object o) {
        int index = o;
        for (Entry e = header.next; e != null; e = e.next) {
                if (o.equals(e.element))
                    return index;
                index++;
        }
        return -1;
    }
```

1. LinkedList 中定义了一个方法，说明该方法在类中正确的方法名称

```
public Object unkown(int index) {
        if (index < 0 || index >= size)
            hrow new IndexOutOfBoundsException("Index:"+index+", Size: " +size);
        Entry e = header.next;
        for (int i = 0; i < index; i++)
                e = e.next;

        return      e.element;
```

}

第七章　队列和栈

一、要求

1. 掌握队列、栈的基本概念：见下
2. 利用单链表实现栈（一般不设立哨兵，单链表的头为栈顶）
3. 利用数组实现栈（栈底为-1）
4. 掌握利用 ArrayList、LinkedList 实现栈
5. 掌握利用双端链表实现队列
6. 掌握利用 ArrayList、LinkedList 实现队列
7. 以 Queue 或 Stack 为例，试述继承与合成的使用方法
8. 掌握栈、队列所有方法的含义及使用

1、队列是元素的序列，在队列中:

(1) 只能在队列尾进行插入;

(2)只能在队列头进行删除、获取和修改.

特点：先进先出；

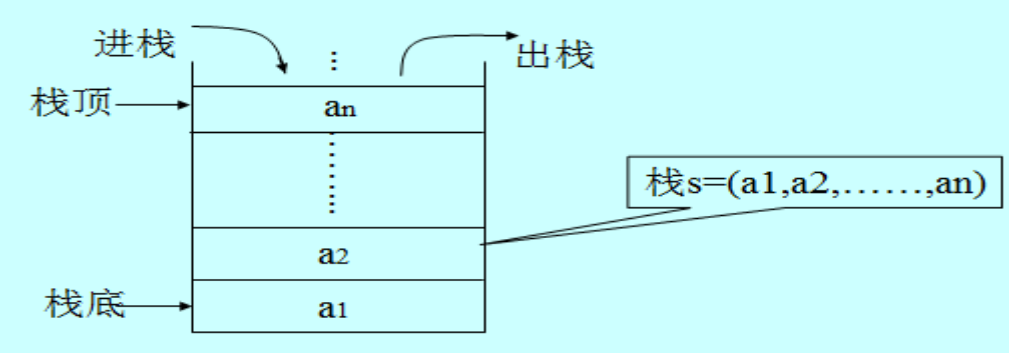
2、栈（stack）

栈的定义和特点

定义：限定仅在表尾进行插入或删除操作的线性表，表尾—栈顶，表头—栈底，(允许插入和删除的一端称为栈顶 (top)，另一端称为栈底(bottom)；

不含元素的空表称空栈；

特点：先进后出（FILO）或后进先出（LIFO）；



3、堆栈 是一个元素序列。堆栈中唯一能被删除、访问或修改的元素是最近插入的元素。这个元素就是位于堆栈顶部的那个元素。

TOP －最近插入的数据元素

PUSH － 入栈，在栈顶插入一个数据元素

POP － 出栈，删除栈顶的数据元素

4、下面操作输入的结果是多少:

```
Stack myStack = new Stack( );
for (int i = 0; i < 10; i++)
      myStack.push (new Integer (i * i));
while (!myStack.isEmpty( ))
      gui.println (myStack.pop( ));
```

结果是：0、1、4、9、16、25、36、49、68、81

二、　　常见题目

1. 教材上的习题、上机题、常用几个方法的时间复杂度

4. 写出下列程序段的输出结果

```
public    static void main(String[] args ){
```

```
        Stack    S = new StackX(10);
        Character    x,y;
        x=new Character（’c’）;
        y= new Character（’k’）;
        S.push(x);
        S. push(new Character（’a’）);
        S.push(y);
        S.pop();
        S.push(new Character（’t’））;
        S.push(x);
        S.pop(x);
        S.push(new Character（’s’））;
        while(!S.isEmpty()){ S.pop(y);System.out.print(y); };
        System.out.println(x);
}
```

5.写出下列程序段的输出结果。

```
 public    static void main(String[] args ){
      Queue Q=new Queue();
      Character x=new Character(’e’), y= new Character(’c’);
      Q.enQueue (new Character(’h’));
      Q.enQueue (new Character(’r’));
      Q. enQueue (new Character(’y’));
      Q. enQueue (x);
      Q. enQueue (x);
      Q.deQueue (x);
      Q. enQueue (new Character(’a’));
      while(!Q.isEmpty()){ Q. deQueue (y);System.out.print(y); };
      System.out.println(x);
}//答：输出为“char”。
```

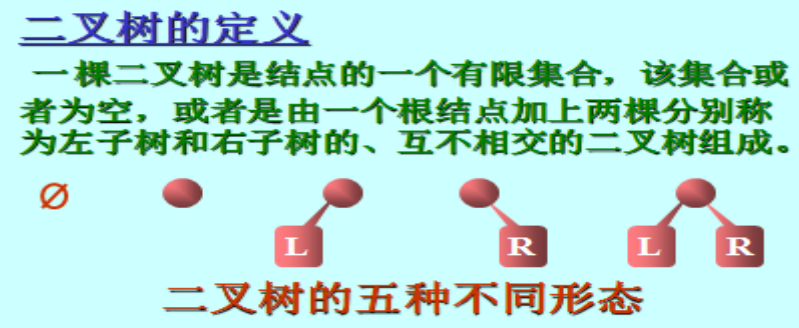
6.说明下面算法的功能

```
      void algo2(Stack S,Object e){
            Stack T = new Stack();
            Object d;
            While(!S.isEmpty())
              {    d=S.pop();
                    if(!d.equals(e)) T.push(d);
               }
              while(!T.isEmpty())
              { d=T.pop();
                 S.push(d);
              }
       }
```

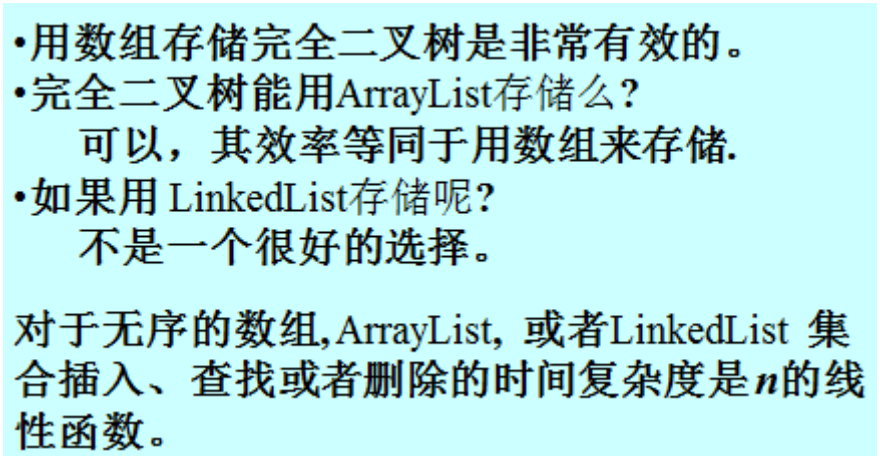
第八章：二叉树与二叉搜索树

一、要求

1. 掌握二叉树的基本概念



2. 熟悉出二叉树的 5 个性质：见下
3. 掌握二叉树的四种遍历方法：先根，中根，后根，宽度优先遍历
4. 掌握二叉搜索树的概念？
5. 掌握二叉树使用链式存储结构时结点描述及二叉树的描述（定义）



6. 掌握二叉搜索树的插入元素算法，并能够依据该算法生成一颗二叉搜索树，会用汉语描述删除元素的算法？见练习
7. 教材中 BinSearchTree 类中查找某一个元素的算法？

8.完全二叉树采用顺序存储结构实现，试描述二叉树的定义及实现最基本方法？

1、树的定义

树是由 n (n 0) 个结点组成的有限集合。如果 n = 0，称为空树；如果 n > 0，则 1）有一个特定的称之为根(root)的结点，它只有直接后继，但没有直接前驱；

2）除根以外的其它结点划分为 m (m>=0) 个 互不相交的有限集合 T0, T1, …, Tm-1，每个集合又是一棵树，并且称之为根的子树。

2、树的特点：每棵子树的根结点有且仅有一个直接前驱，但可以有 0 个或多个直接后继。

3、二叉树的性质：

1）若二叉树的层次从 0 开始，则在二叉树的第 i 层最多有 2i 个结点。(i≥ 0)

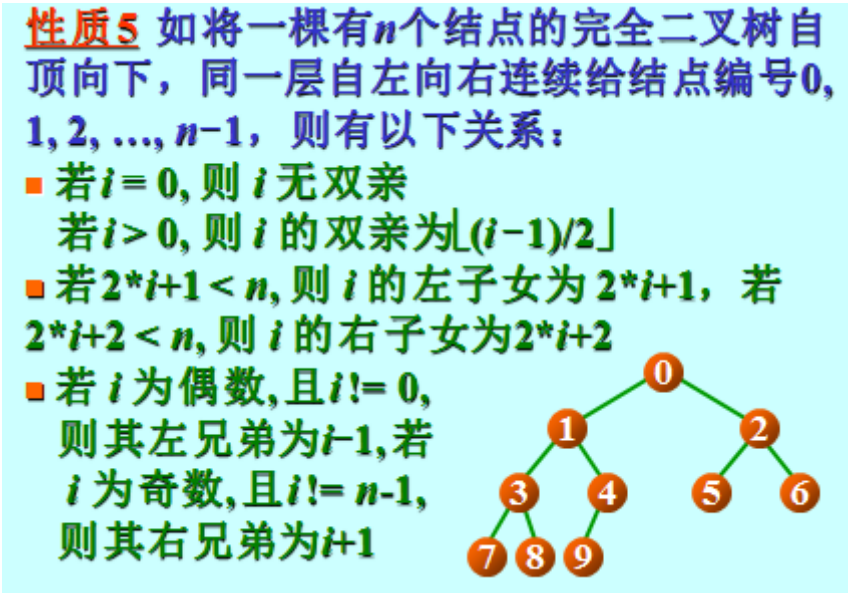
2)高度为 *h* 的二叉树最多有 2*h*+1-1 个结点。(*h* ≥ -1)

3)对任何一棵二叉树，如果其叶结点有 *n*0 个，度为 2 的非叶结点有 *n*2 个，则有：*n*0＝*n*2＋1

若设二叉树的高度为 *h*，则共有 *h*+1 层。

4)具有 *n* (*n* ≥ 0) 个结点的完全二叉树的高度为⌈log2(*n*+1)⌉-1

5）



4、遍历二叉树（见书上）

5、储存结构

BinSearchTree 类不是 Java 集合框架的一部分，但是它实现了 Collection 接口。实际上，BinSearchTree 只实现 Collection 接口的轻微扩充：Set 接口。Set 接口继承了 Collection 接口，并且不允许有重复的值。

6、public boolean contains (Object o) {

```
        Entry temp = root;
         int comp;
        while (temp != null) {
              comp =    ((Comparable)o).compareTo
                                                    (temp.element);
              if (comp == 0)
                    return true;
              if (comp < 0)
                    temp = temp.left;
             else
                    temp = temp.right;
        } // while
        return false;
} // contains
```

7、averageTime(*n*) 是 O(log *n*).

二、 常见题目

1. 教材上的习题、上机题、常用几个方法的时间复杂度
2. 从一棵空的二叉排序树开始，将以下关键码值依次插入：25,13,15,31,7,20,37，请画出插入全部完成后的二叉排序树。
3. 给出下列二叉数的先根、中根、后根遍历的结点序列。
4. 树的基本概念

第十章：TreeMap、TreeSet

一、要求

1. 掌握映射 Map 接口及其的含义？

2. 掌握集合 Set 接口及其的含义？
3. 比较 Collection 接口及其 Set 接口的差别
   Set 对 Coolection 进行了限制，限制了重复的部分。
4. 掌握 TreeMap 数据存储结构，掌握 TreeMap 类内部结点的定义？
5. 掌握 TreeMap 类有常用方法含义及使用方法
6. 使用 TreeMap 类中返回键集、值集、键值集方法，熟悉这些方法如何使用迭代器？

```
package treemappack;
import java.util.*;
public class TestTreeMap {
	public static void main(String[] args) {
		TreeMap tml=new TreeMap();
		/*System.out.println("isEmpty="+tml.isEmpty());*/
		tml.put("20090101", "lisi");
		tml.put("20090203", "wangwu");
		/*System.out.println("put="+tml.put("20090203", "wangwu"));*/
		tml.put("20090102", "zhangsan");
		tml.put("20090104", "hulu");
		/*System.out.println("get="+tml.get("20090203"));
		System.out.println("put="+tml.put("20090203", "wwangwu"));
		System.out.println("get="+tml.get("20090203"));
		System.out.println("size="+tml.size());
		System.out.println("isEmpty="+tml.isEmpty());
		*/

		/*Iterator itr=tml.keySet().iterator();
		while(itr.hasNext())
			System.out.print(" "+itr.next());

		System.out.println();

		Iterator it=tml.values().iterator();
		while(it.hasNext())
			System.out.print(" "+it.next());

		System.out.println();

		Iterator ite=tml.entrySet().iterator();
		while(ite.hasNext())
			System.out.print(" "+ite.next());*/


	TreeMap tm2=new TreeMap();
	tm2.put("20090305", "qqqqq");
	tm2.putAll(tml);
	/*Iterator itr=tm2.keySet().iterator();
		while(itr.hasNext())
			System.out.print(" "+itr.next());

		System.out.println();

		Iterator it=tm2.values().iterator();
		while(it.hasNext())
			System.out.print(" "+it.next());

		System.out.println();

		Iterator ite=tm2.entrySet().iterator();
```

```
                while(ite.hasNext())
                        System.out.print(" "+ite.next());*/
        System.out.println(" "+tm2.remove("20090305"));
                System.out.println(" "+tm2.containsKey("20090305"));
                System.out.println(" "+tm2.containsValue("qqqqq"));
        tm2.clear();
        Iterator itr=tm2.keySet().iterator();
        while(itr.hasNext())
                System.out.print(" "+itr.next());


        }
    }
```

7. 比较 Comparator 接口与 Comparable 接口？练习
8. TreeSet 类有哪些常用方法，解释方法的含义？
9. 比较 TreeMap 与 TreeSet 区别与联系（从用户的角度、从开发人员的角度）？
10. 掌握 TreeMap 与 TreeSet 的简单应用（注意有序）

1、**映射**(MAP)是一种集合,集合中的每个数据元素都由两部分组成: 唯一的键(KEY)部分和值(VALUE)部分。

2、Java 中定义：public interface **Map**

将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。

Map 接口提供三种 *collection 视图*，允许以键集、值集或键-值映射关系集的形式查看某个映射的内容。映射*顺序* 定义为迭代器在映射的 collection 视图上返回其元素的顺序。某些映射实现可明确保证其顺序，如 Tree Map 类；另一些映射实现则不保证顺序，如 Hash Map 类。

3、**用迭代器遍历 students 里的元素:**

```
Iterator itr = students.entrySet( ).iterator( );
while (itr.hasNext( ))
      gui.println (itr.next( ));
```

用迭代器遍历 students 里的键值:

```
Iterator itr = students.keySet( ).iterator( );
while (itr.hasNext( ))
      gui.println (itr.next( ));
```

用迭代器遍历 students 里的值:

```
Iterator itr = students.values( ).iterator( );
while (itr.hasNext( ))
      gui.println (itr.next( ));
```

4、Comparator 接口允许用户在一个类里覆盖此接口. 例如, 如果我们想 String 对象按照字符串长度来排序来替代原来的方法.

例如:

```
public class ByLength implements Comparator {
       public int compare (Object o1, Object o2) {
            return ((String)o1).length( ) – ((String)o2).length( );
 } // compare
 // uses Object's version of equals
} // class ByLength
 TreeMap yourMap = new TreeMap (new ByLength( ));
```

```
yourMap.put (“yes”,    new Integer (1));
yourMap.put (“no”,    new Integer (1));
yourMap.put (“maybe”,    new Integer (1));
yourMap.put (“true”,    new Integer (1));
yourMap.put (“false”,    new Integer (1));
 //现在关键字按长度来排序,其键不允许重复    (即长度相同的).
Iterator itr = yourMap.keySet( ).iterator( );
while (itr.hasNext( ))
 gui.println (itr.next( ));
```

输出结果:

```
no
yes
true
maybe
```

5、由于 thesaurusMap 对象是一棵红黑树，向它的 synonymList 中插入第 i 个元素时，最差情况下，大约需要 log2i。对第 1、2……n 个元素可以用总的花费来作为最大开销时间：nlog2n

6、TreeSet

1）TreeSet 对象是一个不允许元素重复的有序集合。

2）基本上 TreeSet 类的 对象是所有元素都有相同值的 TreeMap 对象。

3）TreeSet 类和 BinSearchTree 类有相同的方法,但是 TreeSet 类比较快, 至少在最坏情况下.对于 add, remove, 和 contains, worstTime(n) log n, BinSearchTree 类是 n 的线性函数.

二、 具体事例

1. 教材上的习题、上机题
2. 请给出下列程序的输出结果：

```
public class ByLength implements Comparator {
      public int compare (Object o1, Object o2) {
            return ((String)o1).length( ) – ((String)o2).length( );
      }
}
public class Test{
      public static void main(String[]args)
      {
            TreeMap yourMap = new TreeMap (new ByLength( ));
            yourMap.put (“fruits”,    new Integer (1));
            yourMap.put (“good”,    new Integer (2));
            yourMap.put (“name”,    new Integer (3));
            yourMap.put (“strutus”,    new Integer (4));
            yourMap.put (“exam”,    new Integer (5));
            yourMap.put (“false”,    new Integer (6));

            Iterator itr = yourMap.keySet( ).iterator( );
            while (itr.hasNext( ))
                System.out.println (itr.next( ));
    }
}
```

第十二章：排序

一、要求

1. 掌握排序的基本概念
2. 掌握常见的排序方法（冒泡、选择、插入）算法及时间复杂度
3. 掌握树排序方法算法及时间复杂度
4. 掌握快速排序方法思路，会画图
5. 掌握 Arrays、Collections 类中排序方法的使用

1、决策树 是一个二叉树，它的每一个非叶子节点代表两个元素的比较，而每个叶子节点代表一个已排序好元素的序列。

左分支: 是

右分支: 否

2、在任意非空二叉树 t 中，叶子节点的数量 leaves (t) <= 2 height (t)

定理：对于任何基于比较的排序, worstTime(n) 不可能少于 O(n log n).

3、**归并排序的基本思路：**

是把待排序的数组分成几个长度几乎相等的子数组，且这些子数组的长度小于 7。接着运用插入排序对两个小规模的子数组中的每一个进行排序，再把两个排序好的子数组归并成两个两倍规模且排好序的子数组。最后，对两个两倍规模的子数组归并，可以得到是开始子数组四倍规模的子数组。这一过程继续执行直到最后把两个规模大小是 n/2 的子数组归并成一个一个排序的规模是 n 的数组。

对于归并排序, worstTime(*n*) 是 O(*n* log *n*)

因此： averageTime(*n*) 是 O(*n* log *n*)

4、树排序

```
public static void treeSort (Object[ ] a) {
        TreeSet tree = new TreeSet ( );
                for (int i = 0; i < a.length; i++)
            tree.add (a [i]);
        tree.toArray (a);
} // method treeSort
```

worstTime(n) 是 O (n log n).

5、堆排序

```
public void heapSort (Object[ ] a) {
        Object temp;
        int length = a.length;

        heap = a;
                size = length;
        for (int i = length / 2 - 1; i >= 0; i--)
            percolateDown (i);
        while (size > 0)
                removeMin( );
        for (int i= 0; i < length / 2; i++) {
                temp = heap [i];
                heap [i] = heap [length - i - 1];
                heap [length - i - 1] = temp;
        } // reversing heap
} // method heapSort
```

第一次循环, worstTime(n) 是 O(n).
对于 while 循环, worstTime(n) IS O(n log n).
对于第二次循环, worstTime(n) IS O(n).
因此，对于 heapSort 方法, worstTime(n) 是多少 ？

6、快速排序

7、二叉树平均高度: O(log n)

log2(n/7) 层；每层至少 n 次比较

总共比较次数 n log2n

averageTime(n) 是 O(n log n).

第十三章：查找和散列类

一、要求

1. 掌握顺序查找、折半查找、二叉搜索树查找算法及时间复杂度、平均查找长度
2. 掌握 Arrays、Collections 类中折半查找方法的使用
3. 掌握 Hash 技术的基本概念（Hash 函数、Hash 表、同义词、冲突、装填因子）
4. 熟悉常见的冲突处理方法（线形探测、二次探测、链地址法），并会按照要求完成 Hash 表的绘制，并会分析平均查找长度
5. 掌握 HashMap 内部存储结构及其定义
6. 掌握 HashMap 类常用方法的使用（特别是 keySet、values、entrySet 方法）
7. 如何理解 hashCode()方法及其与 Hash 函数的关系
8. 掌握 HashMap 的查找算法（containsKey）
9. 掌握 HashSet 常用方法及其使用
10. 掌握 HashSet 实现方式
11. 比较 HashMap、HashSet 区别与联系（从用户的角度、从开发人员的角度）
12. 比较 HashMap、TreeMap 区别与联系（从用户的角度、从开发人员的角度）
13. 比较 HashSet、TreeSet 区别与联系（从用户的角度、从开发人员的角度）

1、顺序查找

```
public boolean contains(Object o) {
 Iterator e = iterator();
 while (e.hasNext())
      if (o.equals(e.next()))
           return true;
 return false;
} // method contains
```

2、二分法检索

注意: 数组必须是有序的.

```
public static int binarySearch(Object[ ] a, Object key) {
 int low = 0;
 int high = a.length-1;

 while (low <= high) {
      int mid =(low + high)/2;
      Object midVal = a[mid];
      int cmp = ((Comparable)midVal).compareTo(key);
      if (cmp < 0)
          low = mid + 1;
      else if (cmp > 0)
```

```
            high = mid - 1;
        else
            return mid; // key found
     } // while
     return -(low + 1);    // key not found
   } // method binarySearch
```

worstTimeU(n) 是log n.

3、RED-BLACK 树查找

```
   private Entry getEntry(Object key) {

     Entry p = root;
     while (p != null) {

        int cmp = compare(key,p.key);
        if (cmp == 0)
            return p;
        else if (cmp < 0)
            p = p.left;
        else
            p = p.right;

     } // while
     return null;

   } // method getEntry
```

worstTimeU(n) 是 n的对数

4、HashMap

averageTimeS(n) 是常数!

```
        HashMap ageMap = new HashMap();

            ageMap.put ("dog", new Integer (15));
            ageMap.put ("cat", new Integer (20));
            ageMap.put ("turtle", new Integer (100));
            ageMap.put ("human", new Integer (75));
            gui.println (ageMap);

        Iterator itr = ageMap.keySet().iterator();
            while (itr.hasNext())
                    gui.println (itr.next());

        itr = ageMap.values().iterator();
            while (itr.hasNext())
                    gui.println (itr.next());
```

下面是输出结果:

```
{cat=20, human=75, dog=15, turtle=100}
cat
human
```

dog
turtle
20
75
15
100

第十四章 图

图的定义和术语

❖ 图(Graph)——图 G 是由两个集合 V(G)和 E(G)组成的,记为 G=(V,E)
其中：V(G)是顶点的非空有限集
E(G)是边的有限集合，边是顶点的无序对或有序对

❖ 有向图——有向图 G 是由两个集合 V(G)和 E(G)组成的
其中：V(G)是顶点的非空有限集
E(G)是有向边（也称弧）的有限集合，弧是顶点的有序对，记为<v,w>，v,w 是顶点，v 为弧尾，w 为弧头

❖ 无向图——无向图 G 是由两个集合 V(G)和 E(G)组成的
其中：V(G)是顶点的非空有限集
E(G)是边的有限集合，边是顶点的无序对，记为（v,w）或（w,v），并且（v,w)=(w,v)

综合：

1. 用户角度：容器（存储一些数据元素）、对该容器使用的方法
2. 数据逻辑结构角度：集合、线性结构、树型结构、图
3. 数据存储结构角度：数组（连续存储）、链式（不连续存储）
4. JDK 中的容器：
   a) 接口
      i. Colletion、Set、List
      ii. Map
   b) 类
      i. TreeSet、HashSet
      ii. ArrayList、LinkedList
      iii. TreeMap、HashMap
   c) 迭代器
      i. 顶层迭代器 Iterator

ii. 各具体类有自己的迭代器（LinkedIterator、TreeIterator、HashIterator）

d) 常见类

i. Arrays（用于数组排序、查找、填充等数组处理）

ii. Collections（用于容器排序、查找、填充等容器处理）

5. 容器使用选用规则

6. JDK 中没有或不好使用的常见数据结构：

a) Stack

b) Queue

c) 非二叉搜索树、树

d) 图