CPT102 学习小组

W7

https://blog.csdn.net/sanmusen_wu/article/details/117375837

正文:

Java和算法

Iterator

Comparable

Collection

List

AbstractList

Array

Bag

Set

Stack

Map

Queue

ArrayList

ArraySet

SortedArraySet

Linked Node(val, nextnode）：

Linked List

Linked Stack

Linked Queue

Tables(Maps)

Hash Tables

Tree

BinaryTree

General Tree

Binary Search Trees

Balanced Search Trees

Graph

一些补充:

Lec04（前中后缀表达式转换，单栈single，双栈dual）：

Lec06（迭代器，比较器）：

Lec13，14（对链表的删除操作的实现步骤）：

Lec16 (给定T(n) 算O(n)):

Lec 21(二叉树左旋LL右旋RR左右旋LR右左旋RL

数据结构:

数据结构是组织和存储数据的方式，使我们能够有效地访问和修改数据。使用合适的数据结构可以提高程序的效率和性能。

1.理解数据结构的两种视角：

数学和逻辑视角（抽象）

代码实现视角（具体）数据结构是数据储存的概念而非编程语言

2.数据结构大致分为线性结构linear 层次结构Hierarchical

数据结构也可分为静态数据结构（数组） 和 动态数据结构（链表 动态数组）

静态和动态各有优缺点

3.数据结构类型：Array和ADT（抽象数据类型）

抽象数据类型：List、 Bag、Set、 Stack、Map、 Queue、Graph、 Tree、Network…

4.数据结构的相关操作：增删改查排序...

工具：Iterator Comparable

Iterator

迭代器是一种设计模式，用于顺序访问集合中的元素，而不需要暴露集合的内部表示。使用迭代器的原因是它提供了一个统一的方法来访问遍历不同类型的数据结构。所有Collection类都可以使用迭代器

· hasNext(): 返回boolean，指针之后是否有元素

· next(): 返回指针之后的下一个元素，移动指针往后移动1个元素

· remove(): 删除上次迭代的元素(这个方法不怎么使用，但是必须要声明)

区分Iterable接口 和 Iterator接口(Lec05):

Comparable

比较器，所有Collection类都可以使用比较器,排序的时候会使用

· compareTo(T ob): Comparable的接口的方法，所有继承自Comparable的对象都要实现该方法，方法代表了natural order，根据传入对象与自己的比较返回整数，自己大返回1，对面大返回-1，平手返回0

· compare(Object，Object): Comparator接口的方法，比较两个对象，<返回-1, =返回0, >返回1

~~natural ordering是什么~~

~~区分Comparable 和 comparator~~

~~Collections.sort(…, Comparator) 的使用~~

Collection/AbstractCollection（Alist）：

Collection

这里讨论java中的接口，所有其他ADT都继承于Collection，`Collection` 接口是Java集合框架的根接口，没有直接的实现，它提供了所有集合共有的基本方法。这个接口主要定义了一组操作集合的基本方法。

· isEmpty(): 返回是否为空

· size(): 返回已包含元素的数量

· contains(E elem): 返回是否包含某个元素

· add(E elem): 在尾部添加一个对象作为元素，返回操作状态

· remove(E elem): 移除一个元素，返回操作状态

· iterator()：返回一个迭代器

· clear(): 利用迭代器清空集合

· addall(Collection): 合并两Collection

· removeAll(Collection): 移除所有位于两Collection交集里的所有元素

· containsAll(Collection): 返回是否包含这个Collection的所有元素

AbstractCollection类

`AbstractCollection` 是实现了 `Collection` 接口的抽象类

AbstractList类

`AbstractList` 是 `List` 接口的一个抽象实现，继承自 `AbstractCollection`，并且添加了针对列表操作的支持。它提供了一些方法的实现，并要求子类至少实现 `get(int index)` 和 `size()` 方法。
声明了size();get(index);等方法
定义了isEmpty();add();contains();clear()等方法

ADT：List、Bag、Set、Stack、Map、Queue

list：

List特性：允许重复，有序，没有访问限制（随机访问）

它是一个接口，常见的实现类有 `ArrayList`、`LinkedList` 等。

List提供了基于索引的访问方式，可以快速地插入、删除和访问元素。

· add(index, item): 在下标处添加元素，该位置以及之后的元素后移一位，返回操作状态

· remove(index):移除下标处元素，该位置之后的元素前移一位，返回操作状态

· get(index): 返回某个元素值下标

· set(index, item): 将下标处元素替换，返回操作状态

· indexOf(item): 返回某个元素下标

· subList(int from, int to): 返回子List

Bag：

Bag特性: 其内元素没有顺序，没有访问限制（随机访问），允许重复

· add(value): 添加元素，返回操作状态

· remove(value): 移除元素，返回操作状态

· contains(value): 返回是否包含该元素

· findElement(value): 返回匹配的元素

什么时候使用Bag：Bag通常用于统计元素的数量，不支持索引访问。

Set：

Set特性：其内元素没有顺序，没有访问限制（随机访问），排除重复项

它主要用于存储不重复的元素集：如HashSet、LinkedHashSet和TreeSet。HashSet是基于哈希表实现的，主要特点是快速查找；

TreeSet是基于红黑树实现的，支持有序的遍历；

LinkedHashSet则维护了插入的顺序。

· add(value): 添加元素，返回操作状态，元素重复则False

· remove(value): 移除元素，返回操作状态

· contains(value): 返回是否包含该元素

· findElement(value): 返回匹配的元素

· clear(): 清空

Stack：

前中后缀转化

Stack特性：其内元素有顺序，先进后出顺序访问，不排除重复项

栈主要支持push（入栈）和pop（出栈）操作。

· push(value): 将对象元素入栈，放置栈顶

· pop(): 出栈一个元素，从栈顶移除

· peel(): 不弹出栈顶元素，返回该栈顶元素

Stack在HTML中的使用

Stack作为栈存储程序各方法的状态

Stack在递归中的使用

利用Stack判断代数表达式是否括号对称

利用Stack处理前缀Polish，中缀Infix，后缀表达式Postfix之间的转换（lec 04）

利用Stack对字符逆序操作

Map：

Map特性：键-值对集合，其内元素无顺序，无访问限制（随机访问），key不能重复，每个元素存储为(key, value)的形式。

常用的实现有 `HashMap` 、 `TreeMap` 和 `LinkedHashMap` 。

HashMap提供了快速的存取能力；

TreeMap保持键的排序；

LinkedHashMap则保持插入顺序。

· get(key): 返回该key对应的val

· put(key, value): 重新设定key对应的val

· remove(key): 移除key为指定key的元素

· containsKey(key)返回是否包含key为指定key的元素

· keySet(): 返回keys的Set对象

· values(): 返回vals的Collection对象

· entrySet(): 返回所有元素的Set对象

使用Map计算文档中单词使用频率（Lec05）

通过key，val，pair分别迭代Map类所有元素

使用Map表示cast表

Queue：

Queue特性：先进先出（FIFO），其内元素有顺序，删除操作只能在一端进行，不排除重复项

`Queue` 接口的实现包括 `LinkedList` 和 `PriorityQueue` 。

PriorityQueue是优先队列，它可以根据比较器或元素的自然顺序来决定元素的出队顺序。

· offer(value): 添加元素进队列入端，返回操作状态，也叫做enqueue

· poll(): 移除并返回出端的一个元素，也叫做dequeue

· peek(): 不移除地返回出端元素

· remove(): 移除并返回出端的元素，当队列为空时抛出异常

· element(): 非移除地返回出端元素，当队列为空时抛出异常

哈夫曼编码Huffman coding的优先级队列应用

Priority Queue是什么，如何用数组实现

使用count的queue

使用front和back指针的循环状queue

上面两者的差别

实现：

语法：

异常（Exception）

异常：是程序在执行过程中发生的异常事件，可由程序控制和捕获。

抛出异常：使用 `throw` 关键字。必要的，因为它可以帮助处理程序中的错误情况，让程序能在问题发生时优雅地恢复或终止。

异常处理：使用 `try` 、 `catch` 和 `finally` 块。 `RuntimeException` （运行时异常）通常是由程序错误导致的，如错误的类型转换；而其他异常（检查型异常）通常是外部错误，如文件未找到。对于不同类型的异常有不同的处理策略。 `getMessage()` 方法用于获取异常描述。

泛型

`<E>` 是一个泛型标记，表示集合中元素的类型。

for/for each

迭代器遍历/for循环遍历-

虽然它们都可以遍历集合中的所有元素，但它们在语法和操作上有一些关键的区别，以及在某些情况下对性能的不同影响。

1. **迭代器遍历**

使用迭代器（Iterator）遍历集合涉及到使用迭代器提供的 `hasNext()` 和 `next()` 方法来访问集合中的元素。

迭代器允许开发者在遍历集合时安全地移除元素，而不会引起 `ConcurrentModificationException` 异常。

优点：

可以在遍历过程中安全地修改集合（如移除元素）。

提供了一种标准的遍历任何实现了 `Iterable` 接口的集合的方式。

缺点：

代码相对冗长。

2. For循环遍历

for循环可以是传统的索引型for循环，也可以是增强型for循环（也称为for-each循环）。增强型for循环内部实际上使用的是迭代器，但语法更简洁。

传统的for循环（索引型）：

```
for (int i = 0; i < list.size(); i++) {
    String item = list.get(i);
    System.out.println(item);
}
```

增强型for循环（for-each）：

```
for (String item : list) {
    System.out.println(item);
}
```

优点：

代码更简洁（尤其是增强型for循环）。

增强型for循环自动处理迭代逻辑，减少编码错误。

缺点：

传统的索引型for循环只适用于可以随机访问元素的集合（如 `ArrayList`），不适合 `LinkedList` 等。

增强型for循环在遍历时不能修改集合（如添加、删除元素），否则会抛出 `ConcurrentModificationException`。

~~性能考虑:~~

~~索引型for循环：在支持快速随机访问的集合（如ArrayList）上表现良好，但在如LinkedList这类需要顺序访问的集合上效率低下，因为每次get(i)都需要从头开始遍历。~~

~~迭代器和增强型for循环：在任何类型的Iterable对象上都能提供稳定的遍历性能，因为它们适应了具体集合的内部迭代机制。~~

总之，选择哪种遍历方式取决于具体需求

在遍历过程中修改集合，迭代器遍历。

追求代码简洁性并且不需要修改集合，增强型for循环

频繁按索引访问元素的情况，传统的for循环

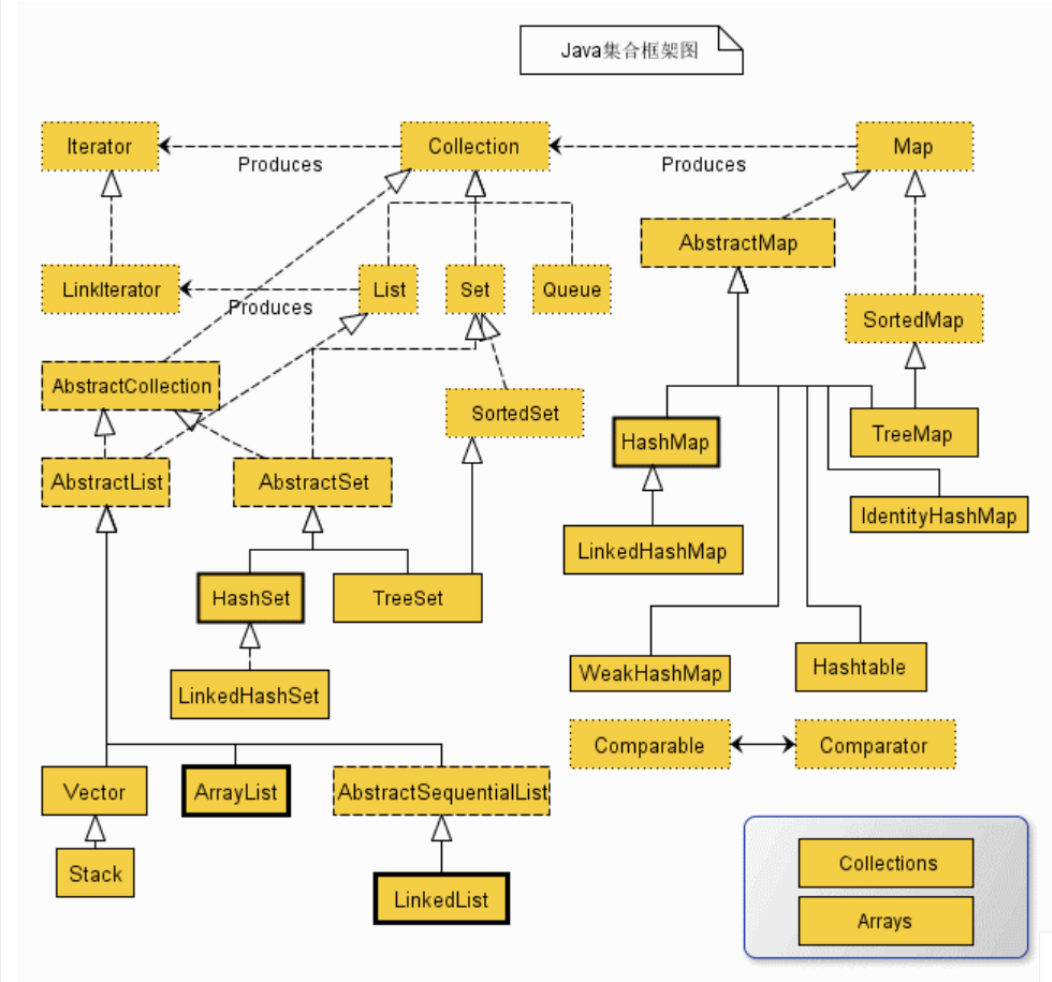
算法：

递归：

递归算法是一种自我调用的过程，用于解决分治法的问题。它将大问题分解为相同类型的更小问题，直到达到易于解决的基案例。递归通常用于如排序、搜索以及在数据结构如树和图中的操作。

BST遍历

W6



ArrayList

AbstractList

ArrayList 继承了AbstractList的接口，并且实现了List接口。 AbstractList作为List的抽象实现，其脱离了具体的数据结构，提供了不同类型的数据结构实现的List所需要的通用方法。

Implementing ArrayList

Initialize the array

使用asList()初始化ArrayList

与数组不同，我们不能直接初始化数组列表。 但是，我们可以使用Arrays类的asList()方法来达到相同的效果。

为了使用asList()方法，我们必须先导入 java.util.Arrays 包。

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
3
4 class Main {
5     public static void main(String[] args) {
```

```
        //创建数组列表
        ArrayList<String> animals = new ArrayList<>(Arrays.asList("Cat",
"Cow", "Dog"));
        System.out.println("ArrayList: " + animals);

        //访问数组列表中的元素
        String element = animals.get(1);
        System.out.println("访问元素: " + element);
    }
}

output:
ArrayList: [Cat, Cow, Dog]
访问元素: Cow
```

method

要随机访问数组列表的元素，我们使用get()方法

```
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        ArrayList<String> animals= new ArrayList<>();

        //在数组列表中添加元素
        animals.add("Dog");
        animals.add("Horse");
        animals.add("Cat");
        System.out.println("ArrayList: " + animals);

        //从数组列表中获取元素
        String str = animals.get(0);
        System.out.print("索引0处的元素: " + str);
    }
}
```

更改Arraylist的元素

```
import java.util.ArrayList;

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

```
        ArrayList<String> animals= new ArrayList<>();
        //在数组列表中添加元素
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Horse");
        System.out.println("ArrayList: " + animals);

        //更改数组列表的元素
        animals.set(2, "Zebra");
        System.out.println("修改后的ArrayList: " + animals);
    }
}
```

删除Arraylist的元素

```
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        ArrayList<String> animals = new ArrayList<>();

        //在数组列表中添加元素
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Horse");
        System.out.println("初始ArrayList: " + animals);

        //从索引2中删除元素
        String str = animals.remove(2);
        System.out.println("最终ArrayList: " + animals);
        System. out.println("删除元素: " + str);
    }
}
```

使用remove all的方法

```
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        ArrayList<String> animals = new ArrayList<>();

        // 在ArrayList中添加元素
```

```
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Horse");
        System.out.println("初始ArrayList: " + animals);

        // 删除所有元素
        animals.removeAll(animals);
        System.out.println("最终ArrayList: " + animals);
    }
}
```

获取ArrayList的长度

要获取数组列表的长度，我们使用size()方法

```
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        ArrayList<String> animals= new ArrayList<>();

        // 在arrayList中添加元素
        animals.add("Dog");
        animals.add("Horse");
        animals.add("Cat");
        System.out.println("ArrayList: " + animals);

        //获取arrayList的大小
        System.out.println("arrayList的大小: " + animals.size());
    }
}

output:
ArrayList: [Dog, Horse, Cat]
arrayList的大小: 3
```

要对数组列表的元素进行排序，可以使用Collections类的sort()方法。

默认情况下，排序以字母或数字升序进行。例如，

```
import java.util.ArrayList;
import java.util.Collections;

class Main {
```

```
    public static void main(String[] args){
        ArrayList<String> animals= new ArrayList<>();

        //在数组列表中添加元素
        animals.add("Horse");
        animals.add("Zebra");
        animals.add("Dog");
        animals.add("Cat");

        System.out.println("未排序的ArrayList: " + animals);

        //对数组列表进行排序
        Collections.sort(animals);
        System.out.println("排序后的ArrayList: " + animals);
    }
}

output:
未排序的ArrayList: [Horse, Zebra, Dog, Cat]
排序后的ArrayList: [Cat, Dog, Horse, Zebra]
```

遍历Arraylist

for循环

```
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        //创建数组列表
        ArrayList<String> animals = new ArrayList<>();
        animals.add("Cow");
        animals.add("Cat");
        animals.add("Dog");
        System.out.println("ArrayList: " + animals);

        //使用for循环
        System.out.println("访问所有元素: ");

        for(int i = 0; i < animals.size(); i++) {
            System.out.print(animals.get(i));
            System.out.print(", ");
        }
    }
}

```

```
output:
ArrayList: [Cow, Cat, Dog]
访问所有元素: Cow, Cat, Dog,
```

For each

```java
import java.util.ArrayList;

class Main {
    public static void main(String[] args) {
        // 创建数组列表
        ArrayList<String> animals = new ArrayList<>();
        animals.add("Cow");
        animals.add("Cat");
        animals.add("Dog");
        System.out.println("ArrayList: " + animals);

        //使用forEach循环
        System.out.println("访问所有元素:  ");
        for(String animal : animals) {
            System.out.print(animal);
            System.out.print(", ");
        }
    }
}


ArrayList: [Cow, Cat, Dog]
访问所有元素:
Cow, Cat, Dog,
```

Ensuring capacity

两个重点：

What are the key issues of implementation when we remove an element from an ArrayList?

delete previous last element去除元素的时候要注意整体数组的位置和长度

What are the key issues of implementation when we add an element from an ArrayList?

Ensure data array has sufficient number of elements to add a new element 要确保有足够的空间去加一个新的元素

Iterator

返回下一个数值return next

去除最后一个 但是只能remove一次 next remove

多个迭代器：

每个迭代器都跟踪自己在列表中的位置

删除返回的最后一个项目是可能的，但是该实现不是智能的，如果对它正在循环访问的 ArrayList 进行任何更改，则可能会损坏。

由于它是一个内部类，因此它可以访问ArrayList 的私有字段

Analysing Cost

Program(benchmarking)

关于cost Algorithms和Program都很重要

- actual programs
- on real machines
- on specific input
- measure elapsed time
- System.currentTimeMillis() → time from system clock in milliseconds (long)
- measure real memory usage

Algorithms

O(1)　常数

O(log n)　对数，情况最多的底数为2（但也可能为其它），但是底数无关紧要，所以不明确说底数

O[(long n)c]　多对数

O(n)　线性

O(n log* n)　log* n为迭代对数

O(n log n)　线性对数

O(n2)　平方

O(nc)，Interger(c>1)　多项式，有时叫“代数”

O(cn)　指数，有时叫“几何”

O(n!)　阶乘，有时叫“组合”

要去寻找最重要的需要计量步数的

重点：只计算复杂度最高的那个（有点像求极限）

需要考量的复杂度：平均复杂度和最坏情况复杂度lec8lec9：

在Java中实现ArrayList类

使用接口、抽象类和具体类来设计数据结构

在Java中实现ArrayList和迭代器的相关知识

操作ArrayList时的成本问题

ArrayList：

ArrayList 类是一个可以动态修改的数组，与普通数组的区别就是它是没有固定大小的限制，我们可以添加或删除元素。ArrayList 继承了 AbstractList ，并实现了 List 接口。

在Java中，我们需要先声明数组的大小，然后才能使用它。一旦声明了数组的大小，就很难更改它。

要解决此问题，我们可以使用ArrayList类。

创建ArrayList（这是我们可以在Java中创建数组列表的方法）

```
1 ArrayList<Type> arrayList= new ArrayList<>();
```

//创建整数类型arraylist
ArrayList<Integer> arrayList = new ArrayList<>();

//创建字符串类型arraylist
ArrayList<StringarrayList = new ArrayList<>();

在上面的程序中，我们使用了Integer和String。 在这里，Integer是int类型的相应包装类。

包装类是包装原始数据类型的类。例如，Integer类包装了int类型，Float类包装了Float类型，等等。

注意：我们不能创建原始数据类型（如int，float，char等）的数组列表。相反，我们必须使用它们对应的包装器类。

对于字符串，String是一个类，没有包装类。因此，我们按原样使用String。

lec10：

分析计算机程序成本（时间和空间成本）

通过算法复杂度分析来评估和优化程序性能

W5：（lec1-6 ttl78）

1. Map的使用

案例：单词频率统计（展示了map存储和操作键值对）

（迭代map的方法）

方法一：迭代器获取key，根据key获取值

方法二：直接迭代器遍历entrySet

```
import java.util.*;

public class Test{
     public static void main(String[] args) {
      Map<String, String> map = new HashMap<String, String>();
      map.put("1", "value1");
      map.put("2", "value2");
      map.put("3", "value3");
      //第一种：普遍使用，二次取值
      System.out.println("通过Map.keySet遍历key和value：");
      for (String key : map.keySet()) {
       System.out.println("key= "+ key + " and value= " + map.get(key));
      }
      //第二种
      System.out.println("通过Map.entrySet使用iterator遍历key和value：");
      Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();
      while (it.hasNext()) {
       Map.Entry<String, String> entry = it.next();
       System.out.println("key= " + entry.getKey() + " and value= " +
entry.getValue());
      }
      //第三种：推荐，尤其是容量大时
      System.out.println("通过Map.entrySet遍历key和value");
      for (Map.Entry<String, String> entry : map.entrySet()) {
       System.out.println("key= " + entry.getKey() + " and value= " +
entry.getValue());
      }
      //第四种
      System.out.println("通过Map.values()遍历所有的value，但不能遍历key");
      for (String v : map.values()) {
       System.out.println("value= " + v);
      }
     }
}
```

（菜鸟资料 用于了解 不用掌握）

Sorted map：key排序好的（升序排列）

Tree map：是sorted map 的实现

2. 迭代器和for each 和for 循环

相对比for循环和for each循环：

for each 循环所需要的代码行数更少

~~2数组时，效率一致~~

~~集合类时，for each循环的遍历本质就是迭代器 iterator的遍历,和普通循环遍历相比，各自有自己适用的场景，~~

~~比如说普通for循环比较适合List类（数组类）遍历通过下标查找数据的，而增强型for循环则比较适合链表结构的集合的遍历。不同数据结构里面的实现方式~~

~~ArrayList时，普通for循环 效率更高~~

~~LinkedList时，for each 效率更高~~

~~（https://blog.csdn.net/qq_24126893/article/details/80437342）~~

For each循环最终会被编译器编译成一个for循环

For each 循环不能完全替代for循环因为他有几个局限性：

~~1.只能对元素顺序访问，~~

2.只能访问所有元素，

3. 当循环中没有index（索引）的时候，无法指定元素进行操作

对于集合中的元素操作，首先明确一点：

集合中的遍历都是有iterator迭代器完成的，for each虽然对集合中的元素操作也会更简便，但是其本质上也是跟for循环一样，用的其实是Iterator，或者可以说只要实现了Iterator的接口，就可以使用for each循环。

当然同样的，for each循环还是有一些限制，在迭代的过程中，除了使用增删器对集合删减元素之外(Iterator.remove())，是不允许直接对集合进行删减操作的否则会出现ConcurrentModificationException的异常

同样对于集合中的for循环，for each循环也还是有数组的for each循环的缺陷，（不能增删，而且不能获取索引）

3. Queue的特性

1Collection of values with an order 有序的

2Constrained access 有访问限制：Only remove from the front

3Two varieties

Ordinary queues：only add at the back

Priority queues 优先队列（有自己的优先级：先取出最优先）

4操作：

`offer(value)` ⇒ 布尔值

向队列添加一个值（有时称为“入队”）。

`poll()` ⇒ 值

移除并返回队列前端的值，如果队列为空则返回null。

（有时称为“出队”，类似于“弹出”pop）。

`peek()` ⇒ 值

返回队列前端的值，如果队列为空则返回null（不从队列中移除）。

`remove()` 和 `element()`

与 `poll()` 和 `peek()` 类似，但如果队列为空会抛出异常。

4. Comparable和 compareTo 的使用，Comparator

Comparable(接口)是用来干什么的？帮助用来自动排序的

ComparaTo（方法）后者ct是前者cb唯一的方法

1、比较者大于被比较者，那么返回正整数

2、比较者等于被比较者，那么返回0

3、比较者小于被比较者，那么返回负整数

Comparator（比较器）

Comparator *可以自定义*实体类对象进行比较，但是Comparable 就只有ComparaTo 这一个方法的比较规则，

5. Exception（异常）

程序在执行过程中，出现的非正常的情况，最终会导致JVM的非正常停止

Catch exception

```
try{可能产生异常的代码}
catch (定义一个异常相关的变量){ 变量的作用就是用来接收程序产生的异常对象
            异常的处理逻辑（想怎么写就怎么写）}
```

1.try中可能产生什么异常对象，catch中就要定义什么异常变量来接收这个异常对象

2.如果try中产生了异常对象，就会执行catch中异常的处理逻辑，执行完catch中的代码，继续执行try catch后边的代码

3.如果try中没有产生异常对象，就不会执行catch中异常的处理逻辑，执行完try中的代码，继续执行try catch后边的代码

4.如果try中产生了多个异常对象，就需要定义多个catch来捕获处理这些异常

异常的类型

（太多了，IOException, IndexOutOfBoundsException，RuntimeException, NullPointerException）ppt上标黑的

除了runtimeException其他都需要处理

```
throws xxxException{
    throw new xxxException("异常信息");
```

1，throws关键字必须写在方法声明处

2，一般在方法内部抛出了什么异常对象，就使用throws关键字在方法上声明抛出什么异常对象给方法的调用者处理

a.在方法内部抛出了多个异常对象，就需要使用throws关键字在方法上声明多个异常对象

b.在方法内部抛出了多个异常对象，有子父类关系，在方法上声明父类异常即可

3，调用一个使用throws关键字声明抛出异常的方法，就必须的处理这个异常对象

1. 可以使用throws关键字继续声明抛出这个异常对象，最终抛出给JVM处理
2. 可以使用try…catch自己手动处理异常

6. Array 和ArrayList

Array可以包含基本类型和对象类型

arraylist只能包含对象类型

区别1存储类型不同

Array:只可存储基本数据类型和对象

ArrayList:只能存储对象

区别2大小不同

Array:被设置为固定大小

ArrayList:是一个可变数组,大小可自动调整

区别3对象所包含的方法不同

Array:所包含的方法没有ArrayList多

ArrayList有很多操作方法:addAll、removeAll、iteration等

7. 泛型

泛型是用来规定数据类型

```
1 public class test {
2 public static class ShowUncheckedWarning {
3 public static void main(String[] args) {
4 java.util.ArrayList<String> list = new java.util.ArrayList<String>();
5 list.add("Java Programming");
6         }
7     }
8 }
```

错误示例：

```
1 public class test {
2 public static class ShowUncheckedWarning {
3 public static void main(String[] args) {
4 java.util.ArrayList list = new java.util.ArrayList();
5 list.add("Java Programming");
6         }}}
```

编译器会有警告

泛型的作用：

一: 类型错误现在就可以在编译时被捕获了，

而不是在运行时当作ClassCastException展示出来。

将类型检查从运行时挪到编译时有助于您更容易找到错误，

并可提高程序的可靠性。

二: 消除强制类型转换(后面将提到)。

泛型的一个附带好处是，消除源代码中的许多强制类型转换。

这使得代码更加可读，并且减少了出错机会。

泛型类

泛型类是指该类使用的参数类型作用于整个类，

即在类的内部任何地方(不包括静态代码区域)

都可把参数类型当做一个真实类型来使用，比如用它做为返回值、

用它定义变量等等。定义泛型类的定义很简单，只需在定义类的时候，

在类名后加入<T>这样一句代码即可，其中T是一个参数，是可变的。

8. BST的前中后（重点）

前序：根左右

中序：左根右

后序：左根中

代码：

后序：

```
void dfs(TreeNode root) {
   dfs(root.left);
   dfs(root.right);
   visit(root);
}
```

中序：

```
void dfs(TreeNode root) {
   dfs(root.left);
```

```
    visit(root);
    dfs(root.right);
}
```

前序：

```
void dfs(TreeNode root) {
    visit(root);
    dfs(root.left);
    dfs(root.right);
}
```

W4：

Lec04 More Collections: Bags, Sets, Stacks, Map

集合库（Collections Library）：

Collection: 最通用的集合接口。Bag

List: 有序集合，可以访问任何位置的元素。Ordered collection

Set: 无序集合，不允许有重复元素。Unordered, no duplicate

Stack: 有序集合，只能在一端添加或移除元素（后进先出）。Ordered collection, limited access

Map: 键值对集合，通过键访问值。Key-value pairs

Queue: 有序集合，只能在一端添加元素，在另一端移除元素。（先进先出）Ordered collection, limited access

Bag（袋子）：

无结构或顺序的集合，允许重复元素。

基本操作包括添加、移除、查找元素，以及获取集合大小等。

Set（集合）：

与Bag类似，但不包含重复元素。

操作包括添加、移除、查找元素等。

Stack（栈）：

按照添加顺序组织的集合，只能在栈顶进行操作。

应用包括处理结构化数据文件、程序执行、撤销操作、表达式求值等。

Map（映射）：

存储键值对的集合，不允许重复的键。

操作包括获取、设置、移除键值对，以及检查键是否存在等。

应用场景包括统计文件中单词的频率、构建直方图等。

栈的应用

处理结构化（嵌套）数据文件。

程序执行，例如处理子任务后返回到上一个任务。

编辑器中的撤销操作。

表达式求值等。

HTML & XML 示例

如何确保Web文档中的XML/HTML标签正确嵌套?

Stack 在Html的应用，因为html 指令是一对对的出现的，并且不可能出现交叉对，所以Stack会很合适

讲解了pop和push

Push就是放在顶部

Pop就是顶出顶部的数据

```
isBalanced = True
stack = []

while isBalanced and not at end of expression:
    nextCharacter = next character in expression

    if nextCharacter is an opening delimiter ('(', '[', or '{'):
        Push nextCharacter onto stack
    elif nextCharacter is a closing delimiter (')', ']', or '}'):
        if stack is empty:
            isBalanced = False
        else:
            openDelimiter = top of stack
            Pop stack
            if openDelimiter and nextCharacter are not a pair of delimiters:
                isBalanced = False

if stack is not empty:
```

```
    isBalanced = False

return isBalanced
```

中缀表达式中的括号、方括号和大括号是否平衡。

将中缀表达式转换为后缀表达式。

a. 创建一个空栈用于存储运算符，和一个空列表用于存储输出的后缀表达式。

b. 从左到右扫描中缀表达式。

c. 如果遇到操作数，将其添加到输出列表。

d. 如果遇到运算符，如 `+`，`-`，`*`，`/` 等，则将其放入栈中。但是，在这之前，需要将栈顶的所有优先级更高或相等的运算符弹出并添加到输出列表。

e. 如果遇到一个左括号 `(`，将其压入栈中。

f. 如果遇到一个右括号 `)`，则将栈中的运算符弹出并添加到输出列表，直到遇到一个左括号为止，然后丢弃这个左括号。

g. 重复步骤2-6，直到输入表达式被完全处理。

h. 如果栈中仍然有运算符，将它们弹出并添加到输出列表。

i. 最后，输出列表就是后缀表达式。

j.

Map（映射）Key-value

存储**键值**对的集合，通过键访问值。

声明和构造时必须指定两种类型：键的类型和值的类型。

Key不重复 get put remove containsKey

使用Map的例子：在文件中查找出现频率最高的单词。

通过Map迭代

如何通过Map迭代？可以通过键、值或键值对集合进行迭代。

总结

介绍了Bag、Set、Stack和Map的应用。和中缀表达式

前三周总结

第一节：数据结构与算法讲座0（CPT102:00 Intro）

8.1 讲师介绍：

Steven Guan：机器学习、大数据分析等领域的研究。

Kok Hoe Wong：企业级IT项目架构和管理经验。

8.2 课程概述：

CPT102课程：数据的组织、访问、管理以及算法的操作。

课程内容包括程序设计基础、数据集合、数据结构实现、算法处理集合、程序分析等。

8.3 学习资源与评估：

8.4 数据集合：

现实生活中的数据集合类型，如书籍、电话簿、学校成绩单等。

8.5 数据结构差异：

有不同类型的数据（different type of Values）

基本数据类型：如整数（int）、浮点数（float）、字符（char）等。

复合数据类型：如结构体（struct）、类（class）、数组（array）等，可以包含多个基本数据类型的组合。

引用数据类型：如链表（linked list）、树（tree）、图（graph）等，通常包含对其他数据结构的引用。

有不同类型数据排列结构（different Structure）

e.g. No structure; Linear structure; Set of key-value pairs; Hierarchical structures; Grid/table

线性结构：数据元素按照线性顺序排列，如数组（Array）、链表（Linked List）、栈（Stack）、队列（Queue）等。

非线性结构：数据元素之间存在复杂的关系，如树（Tree）、图（Graph）等。

无序结构：元素之间没有固定的顺序，如集合（Set）、散列表（Hash Table）等。

有序结构：元素按照某种规则（如大小、时间等）排列，如排序数组、二叉搜索树（BST）等。

有不同的访问修改方式（获取，添加，删除……）

任意位置访问：某些数据结构允许在任何位置进行插入、删除和访问操作，如数组和散列表。

端点访问：在某些数据结构中，只能在两端进行插入和删除操作，如栈和队列。

基于位置的访问：在某些结构中，元素的访问和修改是基于它们在结构中的位置，如数组和矩阵。

基于键值的访问：在映射或散列表中，元素通过键值对进行访问和修改，键用于唯一标识元素。

基于范围的访问：某些数据结构允许基于范围或区间进行访问，如二分搜索树（BST）和区间树（Interval Tree）。

8.6 算法部分

算法示例: 通过制作意大利番茄酱的食谱，比喻了算法的步骤和流程。

算法的终止与程序的不同: 强调了算法必须有终止条件，而程序可能需要持续运行。

我们关注的算法: 介绍了关注点在于创建、访问、操作数据结构的算法，以及它们的成本和性能分析。

算法的应用: 使用Java集合库、设计和创建新的集合类、以及如何高效地添加、删除、搜索、排序等操作。

基本数据结构与算法: 强调了学习基本数据结构和算法的重要性，并提出了衡量算法效率的方法。

算法和程序之间有一些区别：

算法不太看重语法，内容是大于形式的

程序中形式是很重要的，这样才不会让计算机可以识别这些语句

8.7 问答环节:

关于算法必须终止、精确性、指令序列的原因，编写启动IE浏览器和安全关闭计算机的算法。

Why do we insist an algorithm must terminate?

为什么我们坚持算法必须终止?
答：我们坚持算法必须终止是为了确保算法可以在有限的时间内解决问题并给出结果。如果一个算法不终止，那么它可能会无限运行，从而无法得到解决问题的输出，这将导致算法无效。

Why do we insist an algorithm must be precise?

为什么我们坚持算法必须精确?
答：我们坚持算法必须精确是因为算法需要明确无误地描述如何解决特定问题。如果算法不精确，那么可能会导致不确定性，从而无法保证得到正确的结果。

Why are instructions in an algorithm written in a sequence?

为什么算法中的指令要按顺序编写?
答：算法中的指令要按顺序编写是因为算法需要以一种逻辑性的方式一步步解决问题。每一步都依赖于前一步的结果。如果指令顺序混乱，可能会导致算法无法正确执行或解决问题。

Write down an algorithm to start up IE Explorer on a computer.

Input: a computer equipped with Windows which is shut down

Output: a computer up & running with Windows IE Explorer

编写一个算法来启动电脑上的IE浏览器。

输入：一台装有Windows的电脑，已关机

输出：一台运行着Windows IE浏览器的电脑

```
markdownCopy code
1. 按下电脑的电源按钮开机。
2. 等待电脑启动并进入Windows操作系统。
3. 如果有登录界面，输入用户名和密码登录。
4. 等待桌面加载完毕。
5. 双击IE浏览器的图标启动浏览器。
6. 等待IE浏览器打开，算法终止。
```

Write down an algorithm to shutdown a computer safely from Windows.

Input: a computer equipped with Windows which is running under Windows

Output: a computer which is shutdown

编写一个算法安全地从Windows关闭电脑。

输入：一台运行着Windows操作系统的电脑

输出：一台已经关闭的电脑

```
markdownCopy code
1. 关闭所有正在运行的程序和文件。
2. 点击屏幕左下角的“开始”按钮。
3. 在出现的菜单中选择“关机”选项。
4. 等待电脑自动关闭所有程序并安全地关机。
5. 当电脑完全关机后，算法终止。
```

8.8 阅读材料

第二节：数据结构与算法讲座1（CPT102:01 Overview）

1. 动机：

讨论了编写程序和高质量软件的原因，以及如何实现高质量软件设计。

2. 数据结构：

定义:是组织数据的系统方法，以便高效访问。

一种系统的方式去把数据存储成方便访问（操作）的形式（ 不同的数据结构会有不一样的优势和形式，并且每一个数据结构都需要很多种算法去运行其内容，

我们从三个方面学习数据结构：Properties（特征性质）Organization（形式）Operations（操作）

3. 数据结构设计：

讨论了数据结构设计的一般问题，包括数组、链表、栈、队列、映射和树图等。

数据结构设计的一般问题 如何选择和实现适合问题的数据结构。

一维和多维数组 存储数据的线性和矩阵结构。

链表、双向链表及其操作 动态数据结构，允许在任意位置插入和删除元素。

栈及其操作 后进先出的数据结构，常用于实现递归和表达式求值。

队列及其操作 先进先出的数据结构，用于任务调度和缓冲操作。

映射和操作 存储键值对的数据结构，如字典和散列表。

树和图 非线性数据结构，用于表示层次关系和网络。

4. 软件质量：

5. 抽象：

是提取关键信息并忽略非关键信息的过程，有助于简化问题和设计解决方案。

Data Abstraction/Abstract Data Type：关注这个数据代表的是什么，而不是关注怎么去构建他

Procedural Abstraction:把问题看成一个黑盒子，不用关心黑盒里的细节，关注怎么使用这个黑盒子

通过霍夫曼编码的例子展示了抽象的应用。

抽象的过程和实体 提取关键信息并忽略非关键信息的过程，以及由此产生的模型或视图。

软件开发中的抽象 区分数据抽象和过程抽象，确定哪些细节是重要的。

抽象的应用 通过分组和单独处理每个组的细节来设计软件

6. 霍夫曼编码：

压缩算法 数据压缩（一个有趣的小视频）

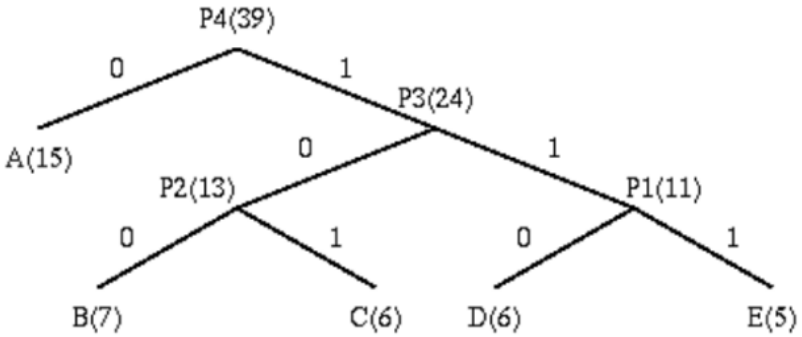
霍夫曼编码的原理 基于数据项出现的频率，使用较少的比特来编码出现频率更高的数据。

向左为0 向右为1

编码和解码 使用霍夫曼编码和解码表进行数据的压缩和解压。

霍夫曼编码的优化性 霍夫曼编码是最优的，因为它最小化了整体的编码长度。

> Huffman tree：
>
> 在编码和解码的时候，我们总是想要用最小的存储来表达最多的东西。在一个或者一串英文单词中，肯定有的字母会用到的更多有的会更少，于是我们就把出现频率高的字母来用较短的编码，出现频率低的就用较长的编码



Symbol	Count	Code	Subtotal (# of bits)
A	15	0	15
B	7	100	21
C	6	101	18
D	6	110	18
E	5	111	15
		TOTAL (# of bits): 87	

List: A(15),B(7),C(6),D(6),E(5)

霍夫曼编码的解码过程涉及以下几个步骤：

a. **构建霍夫曼树**：首先，需要根据原始数据中每个符号的出现频率构建霍夫曼树。这个过程在编码时完成，解码时需要使用相同的霍夫曼树。

b. **接收编码数据**：解码器接收到编码后的数据流，这些数据流是由霍夫曼编码生成的二进制串。

c. **开始解码**：解码器从霍夫曼树的根节点开始，根据接收到的编码数据流中的比特（0或1）向左或向右移动。

d. **遍历霍夫曼树**：每当解码器接收到一个比特，它就会根据这个比特来决定是向左还是向右移动。如果比特是0，解码器就向左移动；如果比特是1，解码器就向右移动。

e. **到达叶节点**：当解码器到达霍夫曼树的一个叶节点时，它就找到了一个完整的编码。这个编码对应于原始数据中的一个数据项。

f. **输出数据项**：解码器输出当前叶节点所代表的数据项，并将指针移回树的根节点，准备开始解码下一个编码。

g. **重复过程**：解码器继续接收编码数据流中的比特，并重复上述过程，直到解码完成所有的数据。

例如，假设我们有一个简单的霍夫曼树，其中字符A、B、C、D和E的编码如下：

```
A: 0
B: 10
C: 110
D: 1110
E: 1111
```

如果编码数据流是 `01101101`，解码过程如下：

从根节点开始。

接收到 `0`，向左移动到节点 `A`。

接收到 `1`，向右移动到节点 `B`。

接收到 `1`，向右移动到节点 `C`。

接收到 `0`，向左移动到节点 `D`。

接收到 `1`，向右移动到节点 `E`。

接收到 `1`，向右移动到节点 `A`（因为A的编码是 `0`，所以这里是一个完整的编码）。

输出字符 `A`，并将指针移回根节点。

继续这个过程，直到解码完成所有的数据。

通过这种方式，解码器可以准确地重建原始数据，即使编码后的比特流没有明确分隔每个数据项。

7. 信息隐藏：

信息隐藏的原则 用户只需要知道模块的关键细节，而不需要了解其内部实现。

信息隐藏的应用 例如，汽车的驾驶界面隐藏了复杂的机械细节，只向司机展示必要的控制元素。

8. 封装：

封装的过程和实体 将数据和方法封装在一个容器中，隔离内部和外部。

面向对象编程中的封装 对象隐藏了内部数据和实现细节，只通过公共接口与外界交互。

9. 空间（存储）和时间（运行 speed of execution）效率

空间效率 选择数据结构时，应尽量减少不必要的内存分配。

时间效率 选择数据结构时，应考虑操作的执行速度，即访问数据项的次数。

10. 静态与动态数据结构

静态数据结构 在创建时大小固定，易于规范，但没有内存分配开销，可能导致容量不足或浪费。

动态数据结构 在运行时可以调整大小，无需预先知道数据项的总数，但有内存分配/释放的开销。

11. 问答环节：

讨论了空间效率和时间效率的评估标准，以及动态数据结构与静态数据结构的一般比较。

True

True

True

Ture

总结：这两节课

1数据结构与算法的基础概念 **数据结构的重要性**

2实际例子（如霍夫曼编码）展示了这些概念的应用

3强调了高质量软件设计的重要性 **抽象、信息隐藏和封装的应用**

4介绍了实现这一目标的关键设计原则 **效率的重要性**

第三节：数据结构与算法讲座2（CPT102:02 Using the Java Collection Libraries）

1. 数据结构编程概述：

介绍了数据结构编程的主题，强调了使用库的重要性，特别是在现代程序是图形用户界面GUI和网络程序）

2. 线性集合编程：

讨论了线性集合的类型：列表（Lists）、集合（Sets）、包（Bags）、映射（Maps）、栈（Stacks）、队列（Queues）和优先队列（Priority Queues）。

介绍了如何使用这些线性集合，包括搜索和排序数据、实现线性集合和排序算法，以及链式数据结构和“指针”。

3. **层次结构集合编程：**

   探讨了层次结构集合的类型，如树、二叉树和普通树。

   讨论了如何使用树结构集合，包括构建树结构、搜索树结构和遍历树结构。

4. **使用库：**

   使用库的重要性，介绍了Java标准库的集合，包括java.util、java.io和javax.swing等。

   如何阅读文档以选择有用的库，导入包或类到程序中，并使用这些类。

5. **标准集合：**

   描述了Java集合库中的接口和类，如Bag、Set、Map、List、Queue、Stack和Graph等。

   抽象数据类型（ADT）的概念，如Set、Bag、Queue、List、Stack和Map等。

6. **Java集合库：**

   详细介绍了Java集合库中的接口，如Collection、List、Set、Queue和Map。

   讨论了Java接口和ADT的关系，以及如何使用Java集合接口。

   （ADT（Abstract Data Type，抽象数据类型）是一种可以表述数据结构的工具，目的是使用户在开发软件时从数学模型的实现细节中抽象出数据结构并进行运算）

7. **参数化类型：**

   集合接口和类的参数化，如何声明特定类型的集合变量。

8. **使用Java集合库：**

   讨论了如何创建接口的实例，以及Java集合库中的类如何实现这些接口。

9. **ArrayList的使用：**

   介绍了ArrayList作为Java集合框架的一部分，包括如何声明ArrayList、使用ArrayList的方法，以及如何操作ArrayList中的元素。

10. **问题与答案：**

    提出了关于线性集合和层次结构集合编程的问题，以及关于软件“库”、Java“包”、Java的IO库和GUI库的问题。

11. **结论：**

总结了我们可以声明变量/字段/参数为某种元素类型的集合，我们可以构造一个新的集合对象，并讨论了我们可以在它们上调用的方法，以及如何迭代集合中的所有元素。

12. 阅读材料：

推荐了相关的阅读材料。

它详细介绍了Java在编程中对数据结构的支持、可创建的数据结构类型、如何在Java中创建和访问数据结构，以及一些实际示例。

第四节：数据结构与算法讲座3（CPT102:03 More on Collections）

Java中集合（Collections）和列表（List）

1. 研究动机：

Java支持的一维数据结构类型

创建和使用这些数据结构

在Java中迭代一维数据结构。

2. ~~代码风格注释：~~

建议省略不必要的 `this.` 前缀，仅在需要时使用；

在只包含一条语句时省略大括号；简化数组赋值操作。

3. 集合类型：

介绍了接口和类的概念，以及接口如何扩展其他接口。举例说明了 `ArrayList` 是 `List` 的实现，而 `List` 又是 `Collection` 的子接口。

4. 集合和列表的方法：

列举了 `Collection` 和 `List` 接口中的一些常用方法。

如 `isEmpty()`，

`size()`，

`contains()`，

`add()`，`add(int index, E elem)`

`remove()`，`remove(int index)`

`iterator()`

`set(int index, E elem)`

`get(int index)` 等，

并对列表特有的方法进行了说明

5. 使用集合类型：

声明和初始化一个集合类型的变量或字段，通过实现类创建对象并调用方法来访问或修改集合。

6. 示例：

提供了一个待办事项列表（TodoList）的例子，从文件中读取任务列表、显示所有任务、添加任务、删除任务以及移动任务到不同位置。

7. 迭代列表：

介绍了如何使用增强型for循环和迭代器（Iterator）来遍历列表中的元素。

8. 迭代器：

解释了增强型for循环背后的工作原理，以及迭代器接口的定义和使用方法。`Scanner`是一个特殊的迭代器。

9. 列表与数组的比较：

列表相对于数组的优势，如没有大小限制、内置方法等。

10. ArrayList的工作原理：

关于`ArrayList`内部存储、大小跟踪、动态扩展和迭代器工作方式的问题。

11. 总结：

回顾了讲义的主要内容，包括集合和列表的使用，以及迭代器的概念。

(迭代器模式，也叫游标模式，是一种提供访问容器对象中各个元素的方法，而不需要暴露容器对象的内部细节。它涉及到遍历算法，可以将遍历方法塞到容器对象中，也可以不提供遍历算法，让使用者自己实现。)

12. 阅读材料：

总结：

关于集合和列表的基础知识，包括它们的创建、使用和迭代方法。实际的编程示例

第五节：数据结构与算法 TUT01（CPT102:01-03）

在IntArrayBag 里面写方法

在IntArrayBagTest 里面写运行命令。

1. `IntArrayBag()`：初始化一个空的包，初始容量为10。
2. `IntArrayBag(int initialCapacity)`：初始化一个指定初始容量的空包。如果初始容量为负数，则抛出异常。
3. `add(int element)`：向包中添加一个新元素。如果新元素会导致包超过其当前容量，则在添加新元素之前增加容量。
4. `addMany(int... elements)`：向包中添加多个新元素。如果新元素会导致包超过其当前容量，则在添加新元素之前增加容量。
5. `addAll(IntArrayBag addend)`：将另一个包的内容添加到此包中。
6. `addAll2(IntArrayBag addend)`：将另一个包的内容添加到此包中，但通过逐个添加元素实现。
7. `clone()`：生成此包的一个副本。
8. `countOccurrences(int target)`：计算此包中特定元素的出现次数。
9. `ensureCapacity(int minimumCapacity)`：更改此包的当前容量。
10. `getCapacity()`：获取此包的当前容量。
11. `remove(int target)`：从包中移除一个指定的元素。
12. `removeMany(int...targets)`：从包中移除多个指定的元素，并返回被移除的元素总数。
13. `size()`：确定此包中的元素数量。
14. `trimToSize()`：将此包的当前容量缩减为其实际大小（即它包含的元素数量）。
15. `union(IntArrayBag b1, IntArrayBag b2)`：创建一个新的包，其中包含两个其他包的所有元素。
16. `intersection(IntArrayBag b1, IntArrayBag b2)`：创建一个新的包，其中包含两个其他包的交集元素。

集合

使用对象可表编程

Part1: 线性集合编程 : (种类) list, sets, Bags, Maps, stacks, Queues, Priority Queues

- 集合编程
- 搜索&排序数据
- 实现线性集合
- 实现排序算法
- 随机数据和"拥有"

值的值以/结构不同

时重复项和访问有不同的规则

抽象 { 不关注元素类型
不关心如何存储或内部架构

Part2: 层次结构集合编程: (种类) 树, 二叉树, 普通树. graph. tree

使用树结构: 构建 Building 搜索 Searching/遍历 Traversing 树结构.

Java.util 集合类及其他实用类
Java.io 输入输出类
Javax.swing 用于 GUI 程序的大型类库
Java.awt

→ 阅读文档, 选择
导入 import
如何使用: 构造函数以创建实例
调用的方法
实现的对象

抽象数据类型: (对象)封装的结构)

ADT是一种在抽象层面上描述的数据类型. { 指定可以对此类型对象执行的操作
指定它将如何表现

eg: Set (操作) add. remove. contain → boolean

⇒ !!

Java 集合库: 接口 Interface { Collection = Bag
List = ordered collection
Set = unordered, no duplicates
Queue : ordered collection, limited access (add at one end, remove from other)
Map = key-value pairs (or mapping)

类 classes { List : ArrayList. LinkedList. Vector ...
Set : HashSet. TreeSet ...
Map : HashMap. TreeMap ...

Java Interface & ADT : Java 接口对应于一个抽象数据类型
指定可以在此类型对象上调用的方法
⇒ !!

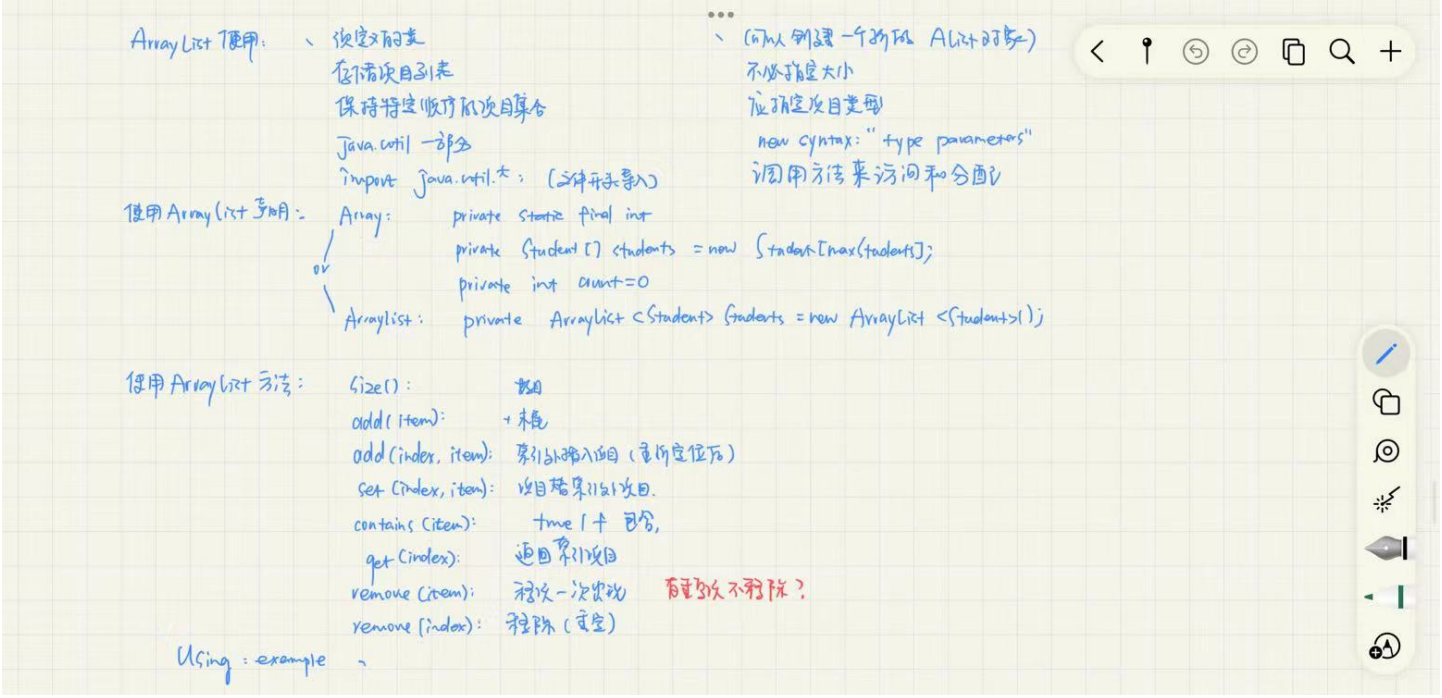
使用 Java 集合接口 : 可以: { 声明一个接口类型的变量、参数、或字段 private List drawing; // a list of Shapes 不用声明!!
在该变量. 参数或字段上调用方法 drawing.add(new Rect(100, 100, 20, 30));

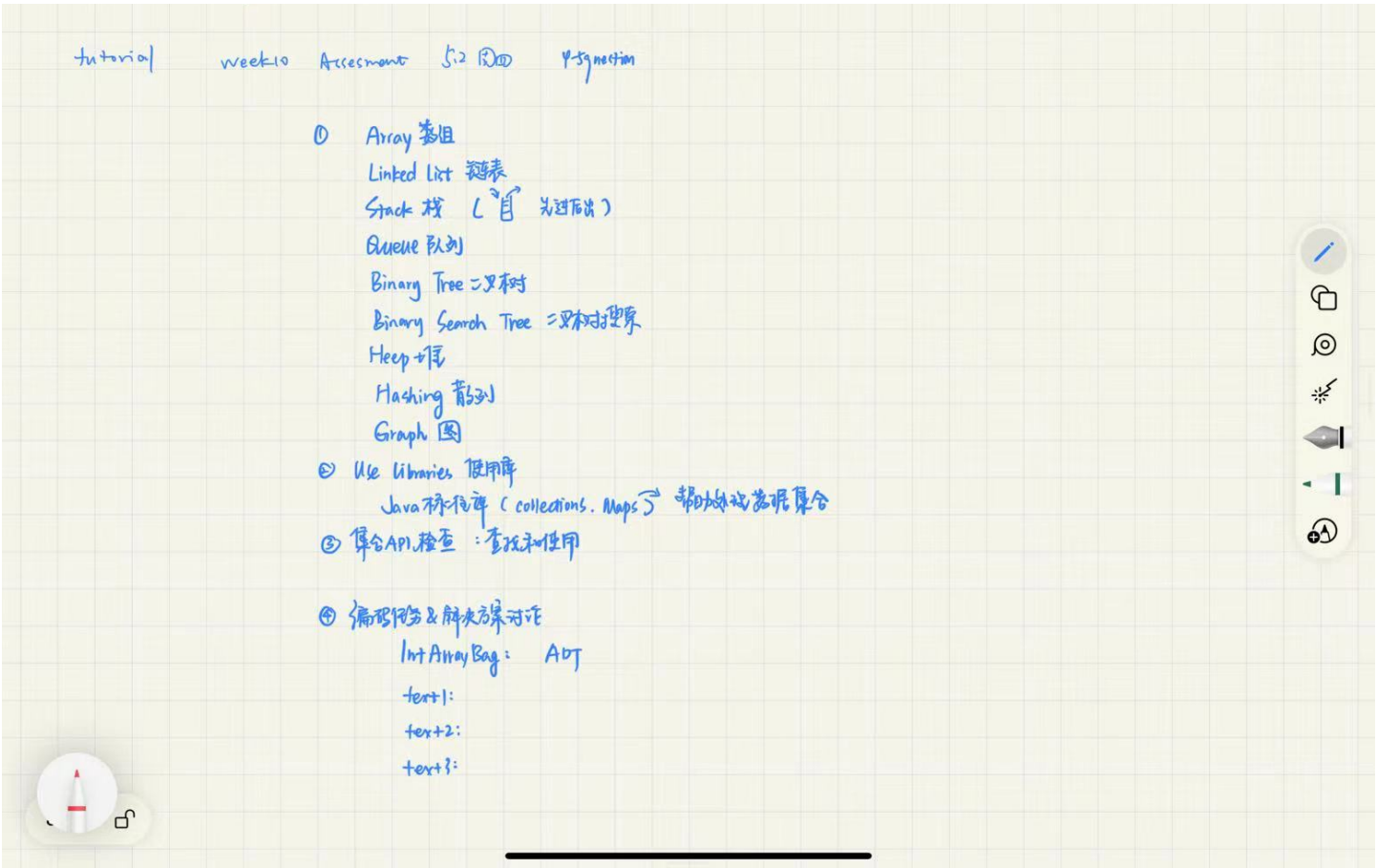
Parameterised Types 参数化类型 : 结构. 访问要求. 值类型相同

声明变量时. 指定对象的实际类型. private Set<Person> friends;

使用Java库 如何创建接口实例? 接口没有 constructors 和构造函数.
~ 定义构造函数 to 构建新实例
定义执行操作的方法体
定义存储值的字段

(eg:) private List<Shape> drawing = new ArrayList<Shape>();
Set<Person> friends = new HashSet<Person>();





如何创建一个可以操作的数据结构

期末考试试卷 两个大题 二分树 暂时还没有对应关系

CPT102 数据结构 时间线与课程信息

概述信息

课程：CPT102-数据结构

学分：5

包含专业：ICS、DMT（不是很确定）

Lec时间：周三1100-1300（W1~6 8~13@SA169）

Tut时间：周四1800-1900（W3~6 10 @SA169）

Lab时间：周五1600-1700（W5~7 10 @SD554）

教师联系方式

注：本课程有两位老师参与Lec教学，学生应在预约时找相应教学内容的讲师提问。

Module Leader

- **Steven Guan**
- 邮件：steven.guan@xjtlu.edu.cn
- Office hour：周三 周四上午10-11@SD425（需要邮件预约）

Teaching Staff

- **Kok Hoe WONG**
- 邮件：kh.wong@xjtlu.edu.cn
- Office hour：@SD431（需要邮件预约）

时间线（请参阅Handbook或W1课件）

Assessments

- **Assessment Task1（10%）**
- **Assessment Task1（10%）**
- **Final exam（80%）**

期末考**期末周**

- 考试形式：开卷（120min）
- 考试内容：**二分搜索树（BST），后序遍历（Post-order Depth First Traversal），二元最小堆（binary min-heap）删除操作。（大题部分）**

详细解释：

1.理解和应用多种数据结构及其内部表示和算法：

数据结构（如*数组*、*链表*、*栈*、*队列*、*树*、*图*、*散列表*等）

内部工作原理，物理和逻辑存储表示。

相关的算法，例如搜索、排序、插入和删除等操作，并理解这些算法的工作原理。

2.能够在不同实现方式之间做出明智的选择，并根据时间和空间复杂度等理由进行选择理由的论证：

解释为什么某种数据结构或算法在特定情况下更为合适，并能够提供合理的论证。

3.能够选择适当的数据结构，以确保算法的高效实现，并能够为之提供理由：

选择最适合的数据结构来实现算法，以提高程序的效率。

Lectures

周数	内容1
1	数据结构概论数据结构 数据类型 抽象数据类型 **Introduction to Data Structures Data Structures, Data Types, and Abstracti**
2	抽象 信息隐藏和封装；霍夫曼码与ADT、优先队列；效率；静态与动态数据结构 **Information Hiding, and Encapsulation; Huffman Codes and ADT, Priority Q vs. Dynamic Data Structures**
3	数据结构编程概述，java库编程，Java集合、对象列表编程、线性集合与层次集合，定义，java数组和数组列表；集合和列表；使用列表和数组列表，遍历列表，迭代器 **Overview of Data Structure Programming, Programming with Java Librarie Programming with Lists of Objects, Linear Collections vs Hierarchical Colle Abstract Datatype definition, Java arrays and ArrayLists; Collections and Li ArrayList, Iterating through List, Iterators, Interfaces vs Classes**
4	包、集、队列、列表、堆栈、地图。各种数据结构的操作，数据结构的应用，递归方理的堆栈，程序堆栈，后缀转换的堆栈，后缀处理的堆栈，用于求值表达式的堆栈，Map的示例，通过Map迭代 **Lists, Stacks, Maps. Operations upon various data structures, Applications Recursive methods, Stack ADT, Stack for web document processing, Progra postfix conversion, Stack for postfix processing, Stack for evaluating expre Interface Specification, Examples of using Map, Iterating through a Map**
5	队列和优先级队列，迭代器与可迭代器对集合进行排序，迭代器vs比较器，比较强v用多个比较器，compareto vs compare **Queues, Iterator vs Iterable; Sorting collections, Iterator vs Comparator, Co Comparable; Sorting with Comparators, Using Multiple Comparators, comp**
6	异常，异常类型，捕获异常，抛出异常，实现集合，抽象类，接口与抽象类与类，列ArrayList，使用数组实现列表，java中的单链表（声明，初始化，使用，迭代）单链表；ArryList；字段和构造函数，ArrayList方法ensureCapacity；ArrayList操作代价间与空间，算法复杂度 **Exc** **Exceptions, Catching exceptions, Throwing Exceptions, Implementing Colle Interfaces vs Abstract Classes vs Classes, Lists Abstract Datatype definition Realisation of lists using arrays; Singly linked lists in Java (declaration, init Data representation of singly linked lists; Immutable List; ArrayList: fields a ArrayList methods; ArrayList: ensureCapacity; Cost of ArrayList operations**

	ArrayList methods; ArrayList: ensureCapacity; Cost of ArrayList operations; operations, Time vs Space, Algorithm complexity
7	
8	递归，递归vs迭代；测试集合；链接结构、实现集合的链接结构；链表操作，内存分点类，使用带有头的链接节点的列表；使用连接节点，创建和便利链表；链接集合方**Recursions, recursion vs iteration; Testing collection implementations; Mot Linked Structures, Linked structures for implementing Collections; linked l allocation, Heap & memory allocation; Linked Node class, List using linked Using Linked Nodes, Creating & Iterating through a linked list; cost of Linke**
9	堆栈/队列及其实现，后进先出，使用带头的链表创建堆栈；先进先出，创建一个队用，用户作业队列，打印假脱队列，1/O事件队列 **Stacks/queues and their implementation; LIFO, Creating a Stack using a Lin FIFO, Creating a Queue using a Linked List with a header; Application of Qu Print spooling queue, I/O event queue**
10	ArrayList操作的代价，Binary Search，Cost of SortedArraySet with Binary Searc并排序；快速排序；桶排序；慢排序vs快排序；分治排序法；各种排序算法的成本 **ArraySet operations; Binary Search, Cost of SortedArraySet with Binary Sea Insertion sort; Merge sort; Quick sort; Bucket sort; slow sorts vs fast sorts; Conquer; Cost of various sorting algorithms**
11	树抽象数据类型定义（树、二叉树）；使用引用或数组实现树；树排序、树遍历、宽归树 **Tr** **definition (trees, binary trees); Realisation of trees using references or arra traversal, Breadth-First traversal; Tree and Recursion, Recursion tree;**
12	搜索列表，（二叉）搜索树；平衡搜索树，AVL树，AVL旋转，AVL高度平衡；树样例现一般树，哈希表，密钥，安全性 **Search trees; Balanced Search Trees, AVL trees, AVL Rotation, AVL height ba & applications; Tree implementation, Implementing Binary Trees, Impleme Hash tables, Hash function, Keys, Security**
13	图论的基本定义；图的性质；路径；树木；有向图及其应用；网络流；连接图；图的和森林；生成树；最小生成树；确定最小生成树的贪心算法；最短路径问题；网络；之间的最大流量 **Basic definitions of graph theory; Properties of graphs; Paths; Trees; Digra applications; network flows; Connected graphs; Incidence matrix and adja Trees and forests; Spanning trees, Minimum spanning tree, Greedy algorith minimum spanning tree; Shortest path problem; Networks; To determine t between two points (source and sink) in a network**