Recursion和OOP面向对象编程

### 理解Recursion

1. **从数列来理解recursion**

Recursion可以看成高中学的抽象数列的一个变种，我们需要做的就是按照要求求出数列的通项公式，还有初始情况

1. **从数学的角度举例**

**Example: Find a recurrence relation for the number of bit strings of length n that contain three consecutive 0's.**

对于bit string，只有0和1两个数，我们先来找通项公式f（n)到底是啥，假设我们现在有n个bit，那么要满足现在有3个连续的0的情况，我们必须知道它有什么样的情况能通过增加一个bit，增加2个bit或者增加3个bit得来

这个时候用倒推法，先考虑对于f(n）来讲最后一位的情况，假设最后一位是1了，那么我们需要在前面的n-1个位置找到连续的3个0，因为f(n-1）根据题目的定义就是n-1长度的时候满足条件的string数量，所以我们可以看成f(n-1)在最后一位是0的时候一定加上其他的某些数能成为f(n）

最后一位是1的情况我们就考虑完成了，那么接下来就是最后一位是0，如果是0，我们保留这个情况（因为这个很明显能往前分析出其他可能性），继续找到倒数第二位，倒数第二位如果是1，我们也必须在前面n-2位中找到连续3个0，也就是f(n-2)，那么如果倒数第二位又是0，我们又要往前找了

倒数第三位如果是1，那么我们就需要满足再前面的f(n-3）的情况，即在n-3长度时候满足题目条件的数量，但是如果倒数第三位是0，那么和我们刚刚说的结合起来就是最后3位都是0，那就是说我前面n-3长度的string可以是任意string，所以我们就找到了它的通项公式：

**f(n) = f(n-1) + f(n-2) + f(n-3) + 2^(n-3)**

然后我们再来找它的initial state，很明显，当string长度是0的时候，f（0） = 0，当string长度是1的时候，f（1）=0，当string长度是2的时候，f（2）=0，从f（3）开始我们才能使用我们的通项公式，那么这样我们就成功地把找到了这样满足条件的结果。

1. **算法题example：**

**赛场上有n支队伍，每一轮会让队伍两两比赛，输的队伍淘汰，没有选中的队伍直接轮空进入下一轮，直到最终选出第一名，那么需要进行几场比赛**

**Test case：**

**n = 7**

**- 1st Round: Teams = 7, Matches = 3, and 4 teams advance.**

**- 2nd Round: Teams = 4, Matches = 2, and 2 teams advance.**

**- 3rd Round: Teams = 2, Matches = 1, and 1 team is declared the winner.**

**Total number of matches = 3 + 2 + 1 = 6.**

**（tip：当n是偶数的时候，都会参赛，也就是一轮会进行n/2场，然后有n/2只队伍晋级，当n是奇数的时候，一支队伍轮空，会举行\_\_\_\_\_\_场比赛（思考一下），会有\_\_\_\_\_支队伍晋级（思考一下））**

**解法一：不使用recursion，使用最基本的loop循环求解**

**思路：我们分成n是奇数和偶数情况，奇数的话，会有(n - 1) / 2 matches, and (n - 1) / 2 + 1 teams advance，偶数的话，就是n/2 matches，然后 n/2 个teams会进入下一轮，sum用来记录每轮增加的新的比赛次数**

public int numberOfMatches(int n) {

int sum = 0;

while(n > 1){

if(n % 2 == 0){

sum = sum + n / 2;

n = n / 2;

}

else{

sum = sum + (n - 1) / 2;

n = (n - 1) / 2 + 1;

}

}

return sum;

}

**解法二：使用recursion**

**思路：假设我们把要写的这个function看成f(n)的话，return的值应该是比赛局数，**

1. **假设n是偶数，那么会有n/2次比赛，然后会有n/2只队伍晋级，那么晋级的队伍又要打比赛，所以下一轮就变成了f（n/2）了，所以：当n是偶数时，f(n) = f(n/2) + n/2**
2. **当n是奇数，那么会有(n - 1) / 2次比赛，然后会有(n - 1) / 2 + 1只队伍晋级，这里加1是因为一只轮空的队伍，那么晋级的队伍又要打比赛，所以下一轮就变成了f（(n - 1) / 2 + 1）了，所以当n是奇数时，f(n) = f（(n - 1) / 2 + 1）+ (n - 1) / 2**

**我们再来看一下初始的情况，n首先没法取负数，而且从上面我们写的情况来看不会出现负数的情况，n是偶数，那么n/2至少是1, （0在这里不可能在定义域里），n是奇数时，(n - 1) / 2 + 1也至少是1，n=2时，其实还是符合上面n是偶数的情况，所以我们就推断出了base case，n=1,当n = 1的时候其实就以为着只剩下一支队伍了，很明显一支队伍没法比赛比的起来，所以当n = 1，那么f(n）应该return 0**

public int numberOfMatches(int n) {

if(n == 1)

return 0;

if (n % 2 == 0)

return numberOfMatches(n/2) + n/2;

else

return numberOfMatches((n - 1) / 2 + 1) + (n - 1) / 2;

}

**解法三：纯数学方面理解：**

**思路：因为每次比赛一定淘汰掉一只队伍，我们初始有n支队伍，需要淘汰掉n-1只队伍才能剩下1只队伍**

public int numberOfMatches(int n) {

return n - 1;

}

1. **Backtracking的理解**

对于backtracking，它更像是一个recursion的进一步暴力枚举，就是先向前列举所有情况，得到一个解或者走不通的时候就回溯，这类的问题非常的困难，并不是能在短暂时间内能彻底讲完的，backtracking的思路主要分为permutation，combination等，就是把题目通过排列组合拆开来，这里仅作一个了解就可以了，基本上backtracking和recursion不同的地方就是recursion是我们找到明确的通项公式，而backtracking是我们需要通过可能成为通项公式的formula一个个往前找，然后再清除期中不一定存在的情况。比较常见的例子为作业中的sudoku。

### OOP面向对象编程：

1. **Class引入**

在java中，基本的类型只有int，char，short等8种，除了这些基本类型，java还设计了很多复杂的class，比如String，ArrayList等等，大部分的数据结构都是通过设计class来实现的，也就是我们在import一个ArrayList，或者LinkedList其实都是在使用着已经写好的class

1. **Class和interface的理解**

首先两个都无法被实例化，如果说class是一个具体的对象，我们可以讲abstract class里面的是一个还不完整的，信息不够的对象，所以无法实例化，而interface是一个规则，作为规则，interface其实是没有constructor的！！！它只有方法，而且是未被定义需要被继承的方法

class可以被继承，继承class我们用extend，我们可以理解为本身父类class是一个模糊概念，继承的class是一个具体概念，比如说父类是ball，子类是football，interface也可以被继承，但是使用的是implements，我们可以理解为众多的class去继承了一个传统的习惯，比如说都可以被序列化，都可以iterate，要实现iteration，就是一种interface的继承

1. **Class和interface的联系与区别**

**相同点：**

1. **interface需要实现，要用implements，而abstract class需要继承，要用extends**
2. **一个类可以实现多个interface，但一个类只能继承一个abstract class**
3. **interface强调特定功能的实现，而abstract class强调所属关系**
4. **尽管interface实现类及abstrct class的子类都必须要实现相应的抽象方法，但实现的形式不同。interface中的每一个方法都是抽象方法，都只是声明的 (declaration, 没有方法体)，实现类必须要实现。而abstract class的子类可以有选择地实现**

**这个选择有两点含义：**

**一是Abastract class中并非所有的方法都是抽象的，只有那些冠有abstract的方法才是抽象的，子类必须实现。那些没有abstract的方法，在Abstrct class中必须定义方法体。**

**二是abstract class的子类在继承它时，对非抽象方法既可以直接继承，也可以覆盖；而对抽象方法，可以选择实现，也可以通过再次声明其方法为抽象的方式，无需实现，留给其子类来实现，但此类必须也声明为抽象类。既是抽象类，当然也不能实例化。**

1. **abstract class是interface与Class的中介。**

**interface是完全抽象的，只能声明方法，而且只能声明pulic的方法，不能声明private及protected的方法，不能定义方法体，也 不能声明实例变量。然而，interface却可以声明常量变量，并且在JDK中不难找出这种例子。但将常量变量放在interface中违背了其作为接 口的作用而存在的宗旨，也混淆了interface与类的不同价值。如果的确需要，可以将其放在相应的abstract class或Class中。**

**abstract class在interface及Class中起到了承上启下的作用。一方面，abstract class是抽象的，可以声明抽象方法，以规范子类必须实现的功能；另一方面，它又可以定义缺省的方法体，供子类直接使用或覆盖。另外，它还可以定义自己 的实例变量，以供子类通过继承来使用。**

下面的表格基本涵盖了主要的两者之间的区别和相同点：

****

1. **Class和interface的设计选择**

假设我们自己在去设计的时候，一般还是用interface会更舒适一些，原因：

1. interface是支持多继承的，我一次性可以继承多个规则，不会像abstract class一样只能extends一个
2. abstract class如果有时定义的太过于明确，继承的class在我们意识到可能要修改某些父类的功能会很麻烦
3. **Abstract class的本质：**

对于abstract class，如果你不写public，系统会默认你位public，因为需要被继承，尽量不要定义static和final的方法，这些是无法被继承的，class如果你写成了final，就无法被继承了，比如说java在写string这个class的时候，string其实就是一个final class，它是不可变的，没有class继承string class，你也无法自己写一个class去继承String class

### 补充1：HW13作业题偶遇内容

**问题引入：**

为什么可以有如下的代码创建：

Queue a = new LinkedList();

**解释：**

当你创建Queue a = new LinkedList()的时候，实际上我们完成的工作是，我们新建了一个Queue的LinkedList对象。就是说，我的type是queue，但是我是使用linkedlist来实现的（linkedlist的结构），所以这意味着这个queue只能使用queue本身的定义的那些method，而不能使用linkedlist里面的其他的method，（queue在java里面是接口，然后linkedlist继承了queue的接口）

其实很多时候你会发现会有其他的class继承了Queue的接口，比如说java中有一个class叫做PriorityQueue，那么我们也可以写成

Queue a = new PriorityQueue();

这个时候我们实现的就是建立一个type为Queue，但是结构确实PriorityQueue的一个对象了，和上面的你写的那个queue来对比的话，他们使用的方法名称虽然一样，但是实现起来会完全不同哦

**补充：**

接口里面只会写abstract method，也就是只有method名称，但是没有具体实现方法，在这里，PriorityQueue和LinkedList同时继承了Queue接口，所以override了原本的那个method，所以这两个class里面有相同的方法名称，但是完全不同的实现方式，当你去创建一个Queue的对象的时候，使用两个LinkedList或者PriorityQueue来创建，就完全不一样了

interface主要作用就是写一个规范让不同的class去继承，superclass用来写一个父类让class去继承 两者都可以用这个方式建立对象：

interface type/superclass type xx = new subclass（）;

### 补充2：HW11作业题偶遇内容

详见subclass文档关于static class和non-static class的讨论