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

2. **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。

3. **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;
    }
```

4. **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

2. **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

3. **Class 和 interface 的联系与区别**

相同点：

A．interface 需要实现，要用 implements，而 abstract class 需要继承，要用 extends
B．一个类可以实现多个 interface，但一个类只能继承一个 abstract class
C．interface 强调特定功能的实现，而 abstract class 强调所属关系
D. 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。

E. 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：

	Abstract class	Interface
实例化	不能	不能
类	一种继承关系，一个类只能使用一次继承关系。可以通过继承多个接口实现多重继承	一个类可以实现多个interface
数据成员	可有自己的	静态的不能被修改即必须是static final，一般不在此定义
方法	可以私有的，非abstract方法，必须实现	不可有私有的，默认是public，abstract 类型
变量	可有私有的，默认是friendly 型，其值可以在子类中重新定义，也可以重新赋值	不可有私有的，默认是public static final 型，且必须给其初值，实现类中不能重新定义，不能改变其值。

4. **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

5. **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