程序：数据处理（数据声明，数据处理）

无论做什么，都是追求高效

程序也如此：高效

模块化编程（高内聚，低耦合）

容易修改，且修改不会造成其他的影响

容易扩展（基于接口的交互，接口和实现分离）

复用，能够减少写重复代码，不去做重复的事，总是能提高我们的效率

面向对象编程思想

封装：模块化的基础，高内聚，低耦合。把相同的放一起（数据，方法）（内聚），只暴露出一些接口（耦合）

多态：满足不同的需求，接口与实现分离，用一个接口就能满足不同的需求

继承：复用，walks like的继承原则。

封装：

核心是能够正确的进行分类，解耦，UI，逻辑， 数据，可能其中各个部分还需要再分，各个功能的逻辑，让各个类的功能单一

目标：高内聚，低耦合（尽量减少对其他东西的依赖）

高内聚：相同的东西尽量放在一起

低耦合：尽量减少对其他东西的依赖，和其他模块的交互尽量的简单

多态：接口与实现相分离

开放接口

封闭实现

继承：

程序设计：

语言：基本的语法：数据声明(类型)，数据处理，关键字(特性)

工具：不同的平台只是不同的工具罢了

实现：具体的设计实现（具体的算法，编程思想，设计思想）

程序设计，是在实践中学会的，不动手是学不会的

工作：

业务：变化的

技术：核心的

业务是慢慢熟悉的（整体熟悉，然后再用到什么熟悉什么），而技术是常年累月积累的，

要成为个技术Hacker，不要成为个逻辑业务狗

多态：相同接口，不同实现。

二次访问，运行时决议

数据是最重要的，数据决定了ui

编程的时候一定要考虑好，要持有哪些数据，要暴露哪些接口

多搭积木，少造轮子，不要做重复的无价值的事

开发应该是基于模块的，基于组件的

No hard code, be config，尽量要是可配置的

代码之间的依赖越少越好

高内聚低耦合，相同的尽量放在一起，这样，修改引起的扰动也小，删除也容易删除

高内聚：相同的尽量放在一起

低耦合：让它的依赖足够的少

相同的的尽量放在一起：数据放在一起，ui放在一起，控制放在一起，逻辑处理放在一起

其中对数据，ui，控制等可能根据情况还需要进一步的再分一次

编程一定要知道耦合在哪里，我们才能去解耦

减小耦合：让它的输入输出是不不依赖其他东西的

复用和可扩展性。

应用场景决定需求，需求决定设计，设计决定实现。

复用：能抽出来复用的就尽量抽出来。复用，减少开发量，将来修改也容易修改

可扩展性：要考虑到以后尽量把接口做好，做全。修改简单

代理是实现多态的一种很好的方式

计算机：算法

经验（多（熟练），广），仿写，看懂写，自己设计写

设计的时候，中间的通道应该是基本不用改的，但出现新需求的时候，我们要改的应该只是配置项而已，要改的只是起点和终点

Code review:能进行代码书写的对比，能让你知道你身边的队友改了什么更了解项目。

动态：灵活性高点，但使用的间接性差

固定的：使用简单，但灵活性差

太复杂的通用是不合理的，而不合理的通用是不合理的

要搞清楚映射关系，才能映射的正确

没有一一对应，映射起来就会很困难

Ui是不应该依赖于数据的（否则数据变化，ui就要变化），而ui是应该有足够的接口来满足数据的

（即在数据中使用ui）接口既要简单，又要功能齐全(封装的足够好)

Mvc

M:model:数据，提供数据

V:view：ui，显示ui，提供足够多的接口

C:control：对ui控制（显示不显示，点击后的反应，反正对ui的控制就应该在这里），根据数据控制ui

根据情况，对各个部分可能还需要再进行划分

应用场景决定需求，需求决定设计设计决定实现。

组合，把它放进来，

继承，就是它，能够进行具体的实现，实现多态。

聚合，把它传进来，聚合属于强依赖的，最好不要用，要用也最好传入的是个基类就行

集成：继承

组合：类里面包含具体的对象，生命周期一定比整体小

聚合：创建的时候需要用到相应的对象，生命周期可以比整体长（一般比整体长）

Callback是个好东西

回调：对执行的结果进行处理

动画的实现应该有个回调

开放封闭原则ocp：指导我们应该进行抽象，该开放的要开放，改封闭的要封闭,

要清楚什么是封闭的，什么是开放的，抽象是封闭的，实现是开放的，

相同的需求绝不相同的实现两次，第二次就应该进行抽象

Lsp：指导我们怎么进行抽象，子类要能代替基类（这样才能说明是一类，是抽象的对）

怎么判断是不是应该继承：Works like， 从属性出发，

比如如果正方形继承与长方形，在利用正方形长宽相等的属性的时候就会挂掉

封装就是把改开放的开放，把该封闭的该封闭，既要好用，又要灵活

是在好用与灵活中做出权衡，接口应该满足所有的需求

发消息的没有错，错在我们把消息处理错啦

能复用的就尽量复用，不要每次都进行创建，这样既节省资源，而且只有一个对象，也容易进行控制

比如打赏荷官的动画，每次都是同样的，只是数值不同，可以只有一个，然后打赏的时候设置数值并显示，然后restart，这样也容易控制，想不显示，就不显示

要让我们的依赖尽量的弱

代理（delegate）：代替处理，其实就是一个指针，说明谁代替它进行处理

（例子：打牌动作条）

怎么给控件进行解耦，做成delegate，这样控件就不需要知道使用它的是什么了，控件就只需要和它自己的代理打交道了

控件应该有足够的接口来满足需求，最好还预留一个接口，来存取数据

控件尽量做成delegate的形式，

即使要聚合，也应该尽量聚合他的基类，而不是派生类

打牌动作条也应该做成一个状态机的形式，在这几种状态之间进行切换，如果在同一状态即使多次调用也没什么问题，这样才是科学的

（例子：Facebook代理）

对一个数据而言，如果它存在多个代理（即使不是同时存在的），我们那就应该有个数组来管理这些代理，谁需要用这些数据，谁就注册进去，谁不用了，谁就把自己删除。自己管自己就可以了，

这样的话我们就不用去担心代理之间切换的顺序了

如果只用一个成员的话，可能会因为之间设置代理的顺序不定，导致需要数据的收不到，比如先设置代理为自己，而另一个又把该指针设置成null了，我自己就收不到数据了，还有就是自己管自己，不会给其他的模块带来影响。

（例子：新手引导的各个步骤）基于行为的组合

抽象一定要抽象的好，抽象出来的应该是尽可能的能够最大复用，让我们不必再去写一些重复代码

比如：新手引导，我们应该再把各个行为抽象出来（下注，加注，看牌。。。）

这样的话，我们就不必每一个步骤中都去实现这些行为

我们只要去配置这些行为就可以啦

再比如：  
对弹框而言，

它的动作有：show，dismiss，点击外部消失，外部区域不可点击，外部区域可点击，一段时间后消失等，我们应该可以组合它的行为

写成函数（其实就相当于一次小的封装）  
1是为了增加可读性，（避免了太过冗长，而且用一个函数名就代替了一段代码的含）

2是为了进行复用