## 为什么要面向对象式开发？

非面向对象开发即面向过程式的开发有什么弊端

1.代码复用性差

2.可扩展性差

3.代码可维护性差

如果我们面向过程式的开发，则牵一发而动全身，那么面向对象的出现可以更好的解决现存的问题。

## 面向对象

技术层面：

-- 高复用：对重复的代码进行封装，提高开发效率。

-- 高扩展：增加新的功能，不修改以前的代码。

-- 高维护：代码可读性好，逻辑清晰，结构规整。

## 三大特征

封装变化——封装

隔离变化——继承

执行变化——多态

#### 封装

——数据角度

将一个对象的基本数据复合成一个自定义类型

就是将某一对象的特性还有行为进行整合，变成一个类



例如：

枪类：

对象——AK47:

数据：

射速，装填子弹的数量

行为：

攻击

——行为角度

对类外提供简单的必要的功能，隐藏实现的细节；



例如：上述枪类，通过我们按（快捷键/界面图标）实现攻击，而使用这个类的人则不需要知道其中的实现原理；

——设计角度

——分而治之

分解需求，建立交互让多个类协同完成

——封装变化

变化点，每个变化点单独做成一个类

——高内聚

类的内部处理一个变化点

——低耦合

类与类的关系，尽量做到互不影响



例如：

老张开车去p城

我们需要满足一个类中只实现一个方法，那么我们就需要定义两个类：

人——数据（name，address） 行为（Go）

车——数据（type，address） 行为（Run）

#### 继承

——定义：重用现有类的概念与功能，并在此基础上进行扩展；

（子类相比父类更加具体）

——作用：隔离客户端代码与实现方式；

（隔离用与做）



例如：

父类：武器

拥有共性——（攻击）

子类：98K，手榴弹，板砖

在拥有父类的所有成员的同时，每个子类都拥有自己的行为和特性

（攻击方式不同），即子类在继承父类的功能上进行自己的拓展；

适用性：

父类和子类在概念上是一致的，且需要进行统一的处理；

#### 多态

——定义：父类的同一种动作或者行为，在不同的子类上有不同的实现；

——作用：通过重写执行不同变化点

（调用父，执行子）



例如：

父类：

枪械（行为：射击——实现细节不同）

子类：

M416 ——射击模式有：单发、全自动

M16A4 ——射击模式有：单发、点射

优点：增强程序扩展性，体现开闭原则；

## 三大关系：

**耦合度**

#### 泛化：

子类与父类的关系，概念的复用；

B类泛化A类，意味B类是A类的一种；

做法：B类继承A类

#### 关联（聚合，组合）：

部分与整体的关系，功能的复用，变化影响一个类；

A与B关联，意味着B是A的一部分；

做法：在A类中包含B类型成员

#### 依赖：

合作关系，一种相对松散的协作，变化影响一个方法；

A类依赖B类，意味A类的某些功能靠B类实现；

做法：B类型作为A类中方法的参数，并不是A的成员

## 六大原则：

#### 开闭原则

对扩展开放，对修改关闭；

增加新功能，不改变原有代码；

#### 类的单一职责

一个类有且只有一个改变它的原因；

#### 依赖倒置

客户端代码(调用的类)尽量依赖(使用)抽象的组件，

抽象的是稳定的，实现是多变的；

#### 组合复用原则

如果仅仅为了代码复用优先选择组合复用，而非继承复用；

（组合的耦合性相对继承低）

#### 里式替换

父类出现的地方可以被子类替换，在替换后依然保持原功能；

子类要拥有父类的所有功能，

子类在重写父类方法时，尽量选择扩展重写，防止改变了功能；

#### 迪米特法则

不要和陌生人说话，

类与类交互时，在满足功能要求的基础上，传递的数据量越少越好

——降低耦合度；



s