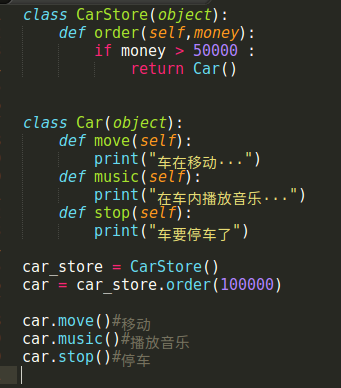
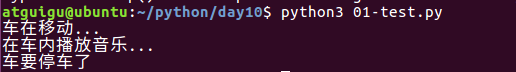
# **1. 面向对象应用--4s店卖车**

## **1.1. 设计一个卖车的4S店，该怎样做呢？**



打印结果



说明

上面的4s店，只能销售那一种类型的车

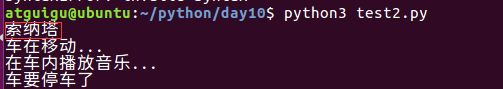
需求：

如果这个是个销售北京现代品牌的车，比如伊兰特、索纳塔(Suonata)、名图(Mintu)等，该怎样做呢？

## **1.2. 设计一个卖北京现代车的4S店**



执行效果：



## 1.3. 想一想--新的需求产生

这样做，不太好，因为当北京现代又生产一种新类型的车时，又得在CarStore类中修改，有没有好的解决办法呢？

# **面向对象应用--设计模式**

## 2.1. 设计模式简介

### 2.1.1. 设计模式的定义

设计模式（Design pattern）代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。

注意：设计模式是高成次的方案，并不关心具体的实现细节，比如算法和数据结构和网页。对应正在尝试解决问题，何种算法和数据结构最优，则是由软件工程师自己把握。

### 2.1.2. 设计模式的常见误区

关于设计模式有一些误解，第一个误解，一开始写代码就应该使用设计模式。我们经常能看到开发人员纠结在代码中应该使用哪种设计模式，他们甚至都还没有尝试一下使用自己的方式解决问题。

这不仅是错误的，而且违背了设计模式的本质，设计模式是在已经有的方案上发现更好的方案（而不是全新发明）,诺你一个方案都没有，又谈何找一个更好的呢？先行动起来，用你的技能尽可能漂亮地解决问题。

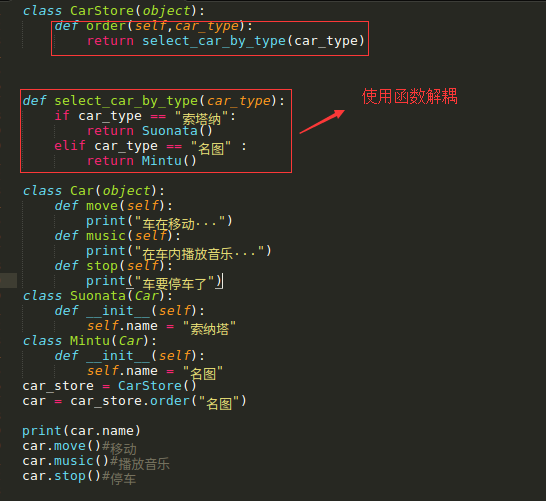
第二个误解是设计模式应随处使用，这会导致方案很复杂，夹杂着多余的接口和分层，而其实往往一个更简单直接的方案就足够了。

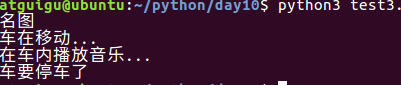
设计模式并不是万能的，仅当代码确实存很烂，难以扩展维护时候，才有使用的必要。

## 2.2. 简单工厂模式

在上一节中，最后留下的个问题，该怎样解决呢？

### **2.2.1. 使用函数实现解耦**





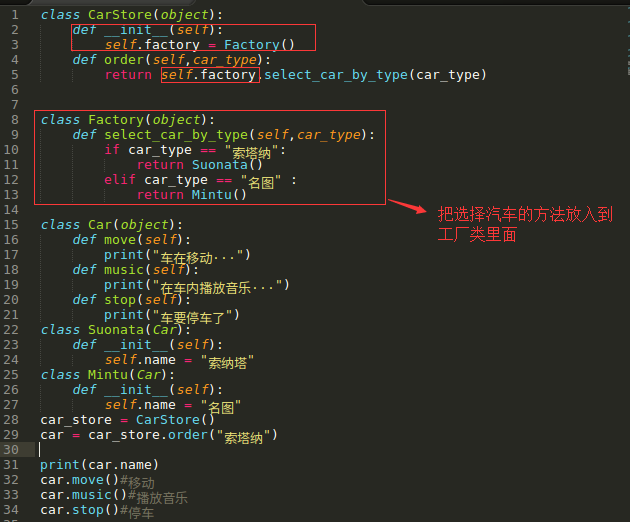
#### 什么是解耦

解耦，就是将程序积木化,就像我们玩的积木一样，各个积木可以组合在一起而形成一个形状，又可以拆分，又可以替换，因为基本上 各个积木块都是独立的，只要他们之间的接口（形状）匹配，就可以灵活地组合在一起,当然，这是理想状态。解耦是在逐渐达到这个理想状态。



### **2.2.2. 使用类来实现**

**简单工厂模式**的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类（这些产品类继承自一个父类或接口）的实例。



接下来很方便就可以添加伊兰特这种车型了

class Yilante(Car):

def \_\_init\_\_(self):

self.name = "伊兰特"

### 2.2.3. **简单工厂模式**

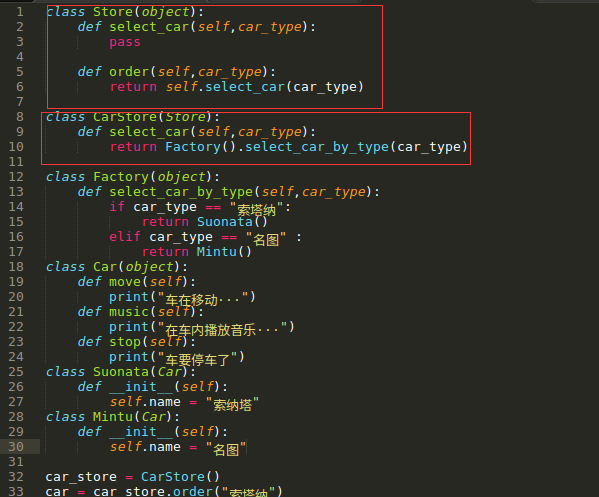
**简单工厂模式**的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类（这些产品类继承自一个父类或接口）的实例。

简单工厂模式的好处：

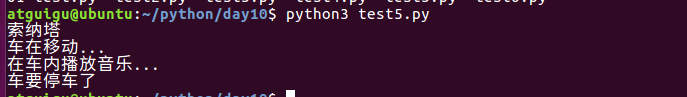
工厂函数、工厂类对具体的生成环节进行了封装，这样有利于代码的后需扩展，即把功能划分的更具体，4s店只负责销售，汽车厂只负责制造。

## 2.3. 工厂方法模式

### 2.3.1. 设置支持买多种品牌-抽取基类



正常运行：

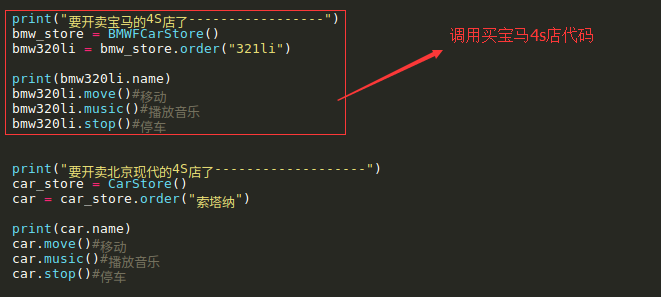


### 2.3.2. 多种品牌的汽车4S店

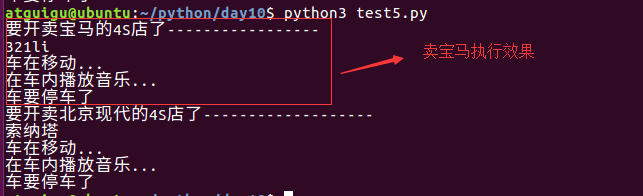
当买车时，有很多种品牌可以选择，比如北京现代、别克、凯迪拉克、特斯拉等，那么此时该怎样进行设计呢？



调用



执行结果



### 2.3.3. 工厂方法模式的定义

定义了一个创建对象的接口(可以理解为函数)，但由子类决定要实例化的类是哪一个，**工厂方法模式**让类的实例化推迟到子类，抽象的CarStore提供了一个选择创建对象的方法select\_car，也叫作**工厂方法**。

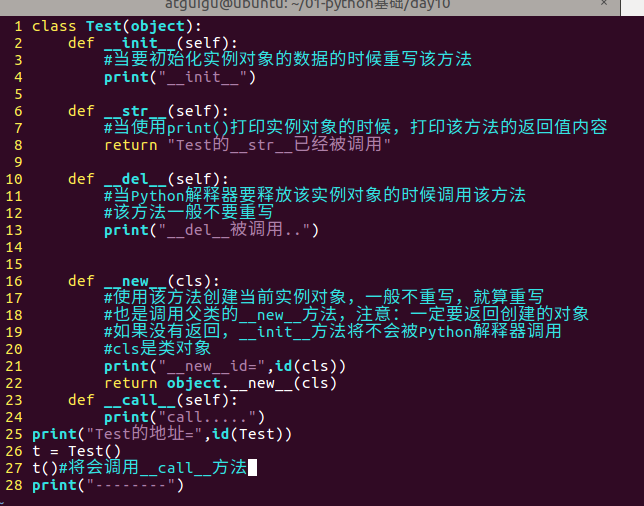
子类真正实现这个select\_car方法创建出具体产品。 创建者类不需要直到实际创建的产品是哪一个，选择了使用了哪个子类，自然也就决定了实际创建的产品是什么。

# \_\_new\_\_方法

object将\_\_new\_\_()方法定义为静态方法，并且至少需要传递一个参数cls，cls表示需要实例化的类，此参数在实例化时由Python解释器自动提供。

## 3.1. \_\_new\_\_和\_\_init\_\_的作用

### 3.1.1. 回顾之前所学的魔法方法



class Test(object):

def \_\_init\_\_(self):

#当要初始化实例对象的数据的时候重写该方法

print("\_\_init\_\_")

def \_\_str\_\_(self):

#当使用print()打印实例对象的时候，打印该方法的返回值内容

return "Test的\_\_str\_\_已经被调用"

def \_\_del\_\_(self):

#当Python解释器要释放该实例对象的时候调用该方法

#该方法一般不要重写

print("\_\_del\_\_被调用..")

def \_\_new\_\_(cls):

#使用该方法创建当前实例对象，一般不重写，就算重写

#也是调用父类的\_\_new\_\_方法，注意：一定要返回创建的对象

#如果没有返回，\_\_init\_\_方法将不会被Python解释器调用

#cls是类对象

print("\_\_new\_\_id=",id(cls))

return object.\_\_new\_\_(cls)

def \_\_call\_\_(self):

print("call.....")

print("Test的地址=",id(Test))

t = Test()

t()#将会调用\_\_call\_\_方法

print("--------")

### 3.1.2 注意和小结

\_\_new\_\_至少要有一个参数cls，代表要实例化的类(类对象)，此参数在实例化时由Python解释器自动提供。

**注意：**

\_\_new\_\_必须要有返回值，返回实例化出来的实例，这点在自己实现\_\_new\_\_时要特别注意，可以return父类\_\_new\_\_出来的实例；

或者直接是object的\_\_new\_\_出来的实例。

\_\_init\_\_有一个参数self，就是这个\_\_new\_\_返回的实例，\_\_init\_\_在\_\_new\_\_的基础上可以完成一些其它初始化的动作，\_\_init\_\_不需要返回值。

我们可以将类比作制造商，\_\_new\_\_方法就是前期的原材料购买环节，\_\_init\_\_方法就是在有原材料的基础上，加工，初始化商品环节

# **4. 单例模式**

## **4.1. 单例是什么**

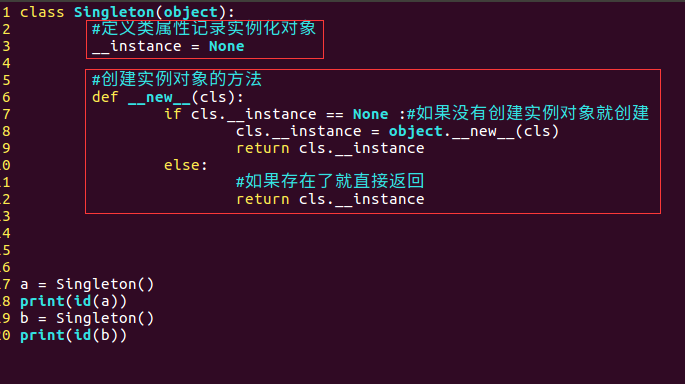
举个常见的单例模式例子，我们日常使用的电脑上都有一个回收站，在整个操作系统中，回收站只能有一个实例，整个系统都使用这个唯一的实例，而且回收站自行提供自己的实例。因此回收站是**单例模式的应用**。

古代的时候，每个朝代只有一个黄帝，这也是单例。

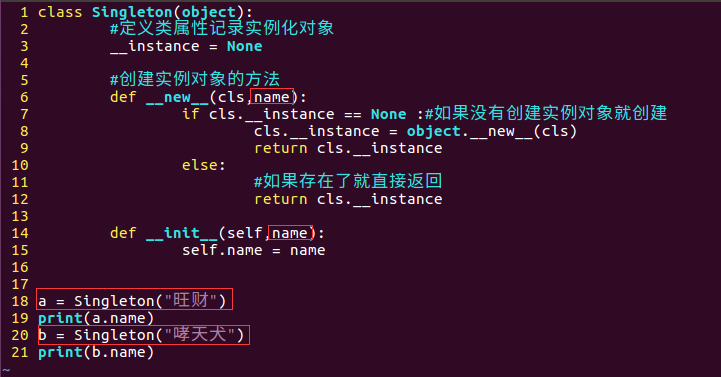
确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，**单例模式**是一种对象创建型模式。

**单例模式**，是一种常用的软件设计模式。在它的核心结构中只包含一个被称为单例的特殊类。通过单例模式可以保证系统中，应用该模式的类一个类只有一个实例。即一个类只有一个对象实例。

## **4.2. 创建单例-保证只有1个对象**



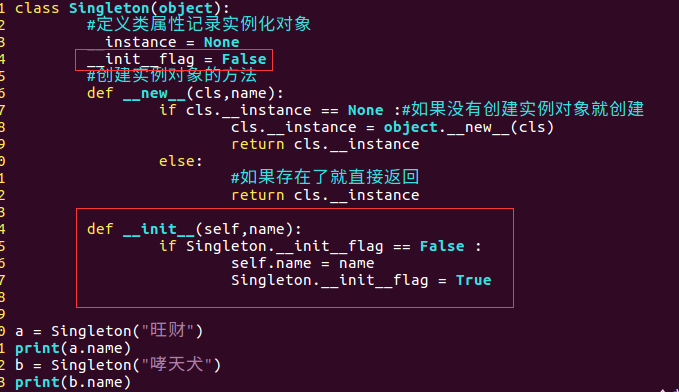
## **4.3. 创建单例时，只执行1次\_\_init\_\_方法**



打印结果



怎么保证打印的名字也是同一个呢？



执行结果：



# class Singleton(object):

\_\_init\_flag = False#False没有赋值name,True已经赋值

\_\_instance = None

#重写\_\_new\_\_方法，创建对象记录下来

#下次创建对象的时候，不去创建新的对象，而是返回已经创建的

def \_\_new\_\_(cls,name):

if cls.\_\_instance == None :

cls.\_\_instance = object.\_\_new\_\_(cls)

print("\_\_new\_\_创建回来的对象的地址==",id(cls.\_\_instance))

return cls.\_\_instance

else :

return cls.\_\_instance

def \_\_init\_\_(self,name):

if Singleton.\_\_init\_flag == False :

self.name = name

Singleton.\_\_init\_flag = True

print("\_\_init\_\_中\_\_new\_\_创建回来的对象的地址==",id(self))

p = Singleton("张三")

print(id(p))

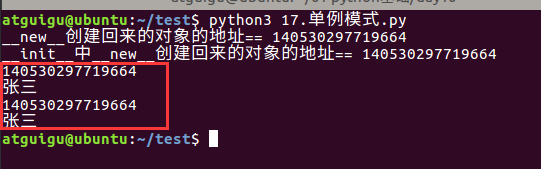
print(p.name)

p2 = Singleton("李四")

print(id(p2))

print(p2.name)

运行结果:



# 5. 代理模式

## 5.1. 什么是代理模式

代理模式的作用是：为其他对象提供一种代理以控制对这个对象的访问。

**代理模式一般涉及到的角色有：**

抽象角色：声明真实对象和代理对象的共同基类(接口)；   
  
代理角色：代理对象角色内部含有对真实对象的引用，从而可以操作真实对象，同时代理对象提供与真实对象相同的基类（接口）以便在任何时刻都能代替真实对象。同时，代理对象可以在执行真实对象操作时，附加其他的操作，相当于对真实对象进行封装。

真实角色：代理角色所代表的真实对象，是我们最终要引用的对象。

## 5.2. 应用场景举例：

比如西门庆找潘金莲，那潘金莲不好意思答复呀，咋办，找那个王婆做代理，表现在程序上时是这样的体现的。

先说说这个场景中的要素：一种类型的女人，潘金莲，王婆，西门庆，后来扩展的贾氏也和西门庆勾上了，我们是假设的，然后西门庆找潘金莲happy,但潘金莲不好意思直接，就找个王婆代理呗。我们看看具体代码。

### 5.2.1. 先定义一种女人

下面是一个基类

class KindWoman(object):

''' 定义一种类型的女人，王婆和潘金莲都属于这个类型的女人

种女人能做什么事情呢？ '''

#抛媚眼

def make\_eyes\_with\_man(self):

pass

#和男人那个...

def happy\_with\_man(self):

pass

### 5.2.2. 一种类型嘛，那肯定是共同特点，定义个潘金莲

被代理类

class PanJinLian(KindWoman):

"定义一个潘金莲是什么样的人 "

def happy\_with\_man(self):

print("潘金莲和男人在做那个...")

def make\_eyes\_with\_man(self):

print("潘金莲抛媚眼...")

### 5.2.3 再定义个丑陋的王婆

代理类

class WangPo(KindWoman):

"""王婆这个人老聪明了，她太老了，

是个男人都看不上她，

但是她有智慧经验呀，

他作为一类女人的代理！

"""

def \_\_init\_\_(self):

self.kind\_woman = PanJinLian()

def set\_kindWoman(self,kindWoman):

'''她可以是KindWomam的任何一个女人的代理，

只要你是这一类型

'''

self.kind\_woman = kindWoman

def happy\_with\_man(self):

'''自己老了，干不了了，但可以叫年轻的代替'''

self.kind\_woman.happy\_with\_man()

def make\_eyes\_with\_man(self):

'''王婆年纪大了，谁看她抛媚眼啊'''

self.kind\_woman.make\_eyes\_with\_man()

### 5.2.4. 两个女主角都上场了，该男主角了，定义个西门庆

class XiMenQiang(object):

'''水浒传是这样写的：西门庆被潘金莲用竹竿敲了一下，

西门庆看痴迷了，被王婆看到了，就开始撮合两人好事，

王婆作为潘金莲的代理人收了不少好处费，

那我们假设一下： \*如果没有王婆在中间牵线，

这两个不要脸的能成事吗？难说得很！'''

def star(self):

wangPo = WangPo()

#然后西门庆说，我要和潘金莲Happy,

# 然后王婆就安排了西门庆丢筷子哪出戏：

wangPo.make\_eyes\_with\_man();

#看到没有表面是王婆在做，其实爽的是潘金莲

wangPo.happy\_with\_man();

XiMenQiang().star()

### 5.2.5 林外一个妹子贾氏登场

那这就是活生生的一个例子，通过代理人实现了某种目的，如果真去了王婆这个中间环节，直接西门庆和潘金莲勾搭，估计很难成就武松杀嫂事件。

     那我们再考虑一下，水浒里面还有没有这类型的女人？有，卢俊义的老婆贾氏（就是和那个管家苟合的那个），这个名字起的：“贾氏”，那我们也让王婆做她的代理：

class JiaShi(KindWoman):

"定义一个贾氏是什么样的人"

def happy\_with\_man(self):

print("贾氏和男人在做那个...");

def make\_eyes\_with\_man(self):

print("贾氏抛媚眼...");

### 5.2.6 西门庆勾潘金莲又勾引贾氏

class XiMenQiang(object):

'''水浒传是这样写的：西门庆被潘金莲用竹竿敲了一下，

西门庆看痴迷了，被王婆看到了，就开始撮合两人好事，

王婆作为潘金莲的代理人收了不少好处费，

那我们假设一下： \*如果没有王婆在中间牵线，

这两个不要脸的能成事吗？难说得很！'''

def star(self):

wangPo = WangPo()

#然后西门庆说，我要和潘金莲Happy,

# 然后王婆就安排了西门庆丢筷子哪出戏：

wangPo.make\_eyes\_with\_man();

#看到没有表面是王婆在做，其实爽的是潘金莲

wangPo.happy\_with\_man();

#王婆成了贾氏的代理了

jiashi = JiaShi()

wangPo.set\_kindWoman(jiashi)

wangPo.make\_eyes\_with\_man();

#看到没有表面是王婆在做，其实爽的是潘金莲

wangPo.happy\_with\_man();

XiMenQiang().star()

说完这个故事，那我总结一下，代理模式主要使用了Python的多态和继承，干活的是被代理类，代理类主要是接活，你让我干活，好，我交给幕后的类去干，你满意就成，那怎么知道被代理类能不能干呢？同根就成，大家知根知底，你能做啥，我能做啥都清楚得很，同样一个（基类）接口呗。好了不多说了，慢慢体会吧。

## 5.3. 完整代码

class KindWoman(object):

''' 定义一种类型的女人，王婆和潘金莲都属于这个类型的女人

种女人能做什么事情呢？ '''

#抛媚眼

def make\_eyes\_with\_man(self):

pass

#和男人那个...

def happy\_with\_man(self):

pass

class PanJinLian(KindWoman):

"定义一个潘金莲是什么样的人 "

def happy\_with\_man(self):

print("潘金莲和男人在做那个...")

def make\_eyes\_with\_man(self):

print("潘金莲抛媚眼...")

class WangPo(KindWoman):

"""王婆这个人老聪明了，她太老了，

是个男人都看不上她，

但是她有智慧经验呀，

他作为一类女人的代理！

"""

def \_\_init\_\_(self):

self.kind\_woman = PanJinLian()

def set\_kindWoman(self,kindWoman):

'''她可以是KindWomam的任何一个女人的代理，

只要你是这一类型

'''

self.kind\_woman = kindWoman

def happy\_with\_man(self):

'''自己老了，干不了了，但可以叫年轻的代替'''

self.kind\_woman.happy\_with\_man()

def make\_eyes\_with\_man(self):

'''王婆年纪大了，谁看她抛媚眼啊'''

self.kind\_woman.make\_eyes\_with\_man()

class JiaShi(KindWoman):

"定义一个贾氏是什么样的人"

def happy\_with\_man(self):

print("贾氏和男人在做那个...");

def make\_eyes\_with\_man(self):

print("贾氏抛媚眼...");

class XiMenQiang(object):

'''水浒传是这样写的：西门庆被潘金莲用竹竿敲了一下，

西门庆看痴迷了，被王婆看到了，就开始撮合两人好事，

王婆作为潘金莲的代理人收了不少好处费，

那我们假设一下： \*如果没有王婆在中间牵线，

这两个不要脸的能成事吗？难说得很！'''

def star(self):

wangPo = WangPo()

#然后西门庆说，我要和潘金莲Happy,

# 然后王婆就安排了西门庆丢筷子哪出戏：

wangPo.make\_eyes\_with\_man();

#看到没有表面是王婆在做，其实爽的是潘金莲

wangPo.happy\_with\_man();

#王婆成了贾氏的代理了

jiashi = JiaShi()

wangPo.set\_kindWoman(jiashi)

wangPo.make\_eyes\_with\_man();

#看到没有表面是王婆在做，其实爽的是潘金莲

wangPo.happy\_with\_man();

XiMenQiang().star()

# 6. 面向对象应用-射杀游戏

## 6.1. 各个类有的属性和方法



什么方法和什么属性适合封装在那个类中



### 6.1.1. 人类

#### 属性

姓名 nem

血量 hp

持有的枪 gun

#### 方法

安装子弹

安装弹夹

拿枪（持有抢）

开枪

### 6.1.2. 枪类

#### 属性

弹夹（默认没有弹夹，需要安装）

#### 方法

连接弹夹（保存弹夹）

射子弹

### 6.1.3. 子弹类

#### 属性

杀伤力

#### 方法

伤害敌人(让敌人掉血)

### 6.1.4. 弹夹类

#### 属性

容量（子弹存储的最大值20）

当前保存的子弹

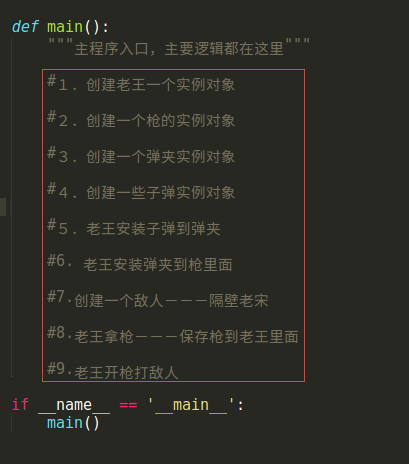
#### 方法

保存子弹（安装子弹的时候）

弹出子弹（开枪的时候）

## 6.2. 开发顺序

### 6.2.1. 框架搭建



def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

#２．创建一个枪的实例对象

#３．创建一个弹夹实例对象

#４．创建一些子弹实例对象

#５．老王安装子弹到弹夹

#6.　老王安装弹夹到枪里面

#7.创建一个敌人－－－隔壁老宋

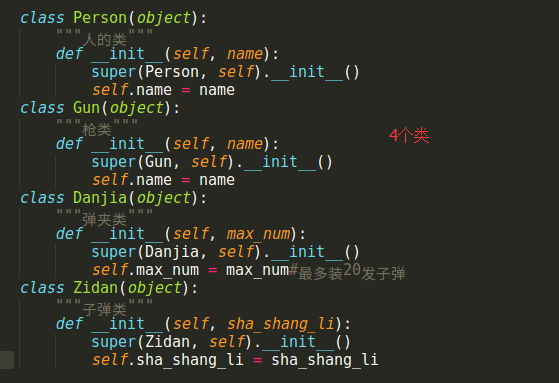
#8.老王拿枪－－－保存枪到老王里面

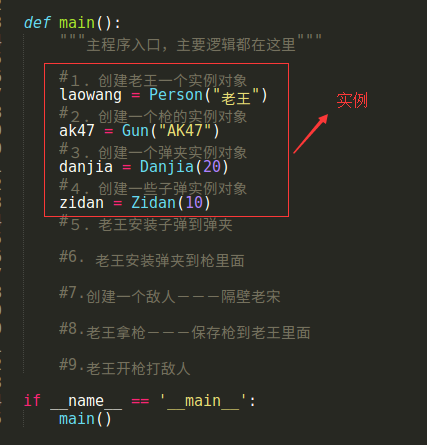
#9.老王开枪打敌人

if \_\_name\_\_ == '\_\_main\_\_':

main()

### 6.2.2. 设计类、创建实例对象





class Person(object):

"""人的类"""

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

class Gun(object):

"""枪类"""

def \_\_init\_\_(self, name):

super(Gun, self).\_\_init\_\_()

self.name = name

class Danjia(object):

"""弹夹类"""

def \_\_init\_\_(self, max\_num):

super(Danjia, self).\_\_init\_\_()

self.max\_num = max\_num#最多装20发子弹

class Zidan(object):

"""子弹类"""

def \_\_init\_\_(self, sha\_shang\_li):

super(Zidan, self).\_\_init\_\_()

self.sha\_shang\_li = sha\_shang\_li

def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

laowang = Person("老王")

#２．创建一个枪的实例对象

ak47 = Gun("AK47")

#３．创建一个弹夹实例对象

danjia = Danjia(20)

#４．创建一些子弹实例对象

zidan = Zidan(10)

#５．老王安装子弹到弹夹

#6.　老王安装弹夹到枪里面

#7.创建一个敌人－－－隔壁老宋

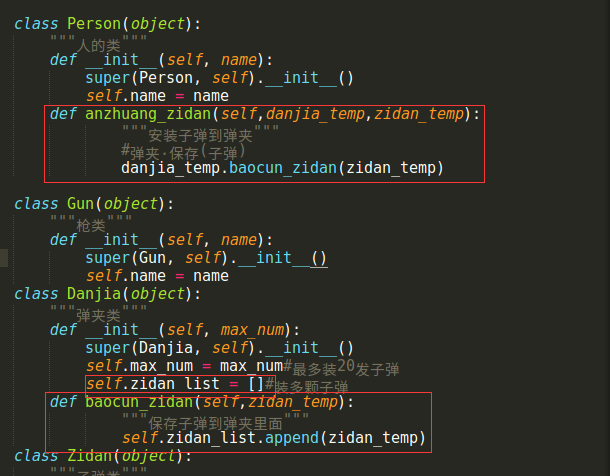
#8.老王拿枪－－－保存枪到老王里面

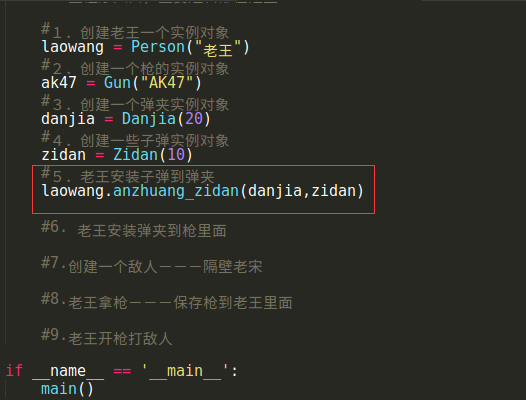
#9.老王开枪打敌人

if \_\_name\_\_ == '\_\_main\_\_':

main()

### 6.2.3. 安装子弹





class Person(object):

"""人的类"""

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

def anzhuang\_zidan(self,danjia\_temp,zidan\_temp):

"""安装子弹到弹夹"""

#弹夹.保存(子弹)

danjia\_temp.baocun\_zidan(zidan\_temp)

class Gun(object):

"""枪类"""

def \_\_init\_\_(self, name):

super(Gun, self).\_\_init\_\_()

self.name = name

class Danjia(object):

"""弹夹类"""

def \_\_init\_\_(self, max\_num):

super(Danjia, self).\_\_init\_\_()

self.max\_num = max\_num#最多装20发子弹

self.zidan\_list = []#装多颗子弹

def baocun\_zidan(self,zidan\_temp):

"""保存子弹到弹夹里面"""

self.zidan\_list.append(zidan\_temp)

class Zidan(object):

"""子弹类"""

def \_\_init\_\_(self, sha\_shang\_li):

super(Zidan, self).\_\_init\_\_()

self.sha\_shang\_li = sha\_shang\_li

def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

laowang = Person("老王")

#２．创建一个枪的实例对象

ak47 = Gun("AK47")

#３．创建一个弹夹实例对象

danjia = Danjia(20)

#４．创建一些子弹实例对象

zidan = Zidan(10)

#５．老王安装子弹到弹夹

laowang.anzhuang\_zidan(danjia,zidan)

#6.　老王安装弹夹到枪里面

#7.创建一个敌人－－－隔壁老宋

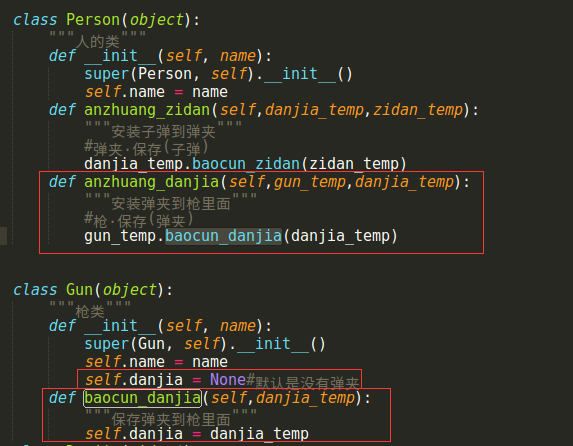
#8.老王拿枪－－－保存枪到老王里面

#9.老王开枪打敌人

if \_\_name\_\_ == '\_\_main\_\_':

main()

### 6.2.4. 安装弹夹





class Person(object):

"""人的类"""

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

def anzhuang\_zidan(self,danjia\_temp,zidan\_temp):

"""安装子弹到弹夹"""

#弹夹.保存(子弹)

danjia\_temp.baocun\_zidan(zidan\_temp)

def anzhuang\_danjia(self,gun\_temp,danjia\_temp):

"""安装弹夹到枪里面"""

#枪.保存(弹夹)

gun\_temp.baocun\_danjia(danjia\_temp)

class Gun(object):

"""枪类"""

def \_\_init\_\_(self, name):

super(Gun, self).\_\_init\_\_()

self.name = name

self.danjia = None#默认是没有弹夹

def baocun\_danjia(self,danjia\_temp):

"""保存弹夹到枪里面"""

self.danjia = danjia\_temp

class Danjia(object):

"""弹夹类"""

def \_\_init\_\_(self, max\_num):

super(Danjia, self).\_\_init\_\_()

self.max\_num = max\_num#最多装20发子弹

self.zidan\_list = []#装多颗子弹

def baocun\_zidan(self,zidan\_temp):

"""保存子弹到弹夹里面"""

self.zidan\_list.append(zidan\_temp)

class Zidan(object):

"""子弹类"""

def \_\_init\_\_(self, sha\_shang\_li):

super(Zidan, self).\_\_init\_\_()

self.sha\_shang\_li = sha\_shang\_li

def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

laowang = Person("老王")

#２．创建一个枪的实例对象

ak47 = Gun("AK47")

#３．创建一个弹夹实例对象

danjia = Danjia(20)

#４．创建一些子弹实例对象

zidan = Zidan(10)

#５．老王安装子弹到弹夹

laowang.anzhuang\_zidan(danjia,zidan)

#6.　老王安装弹夹到枪里面

laowang.anzhuang\_danjia(ak47,danjia)

#7.创建一个敌人－－－隔壁老宋

#8.老王拿枪－－－保存枪到老王里面

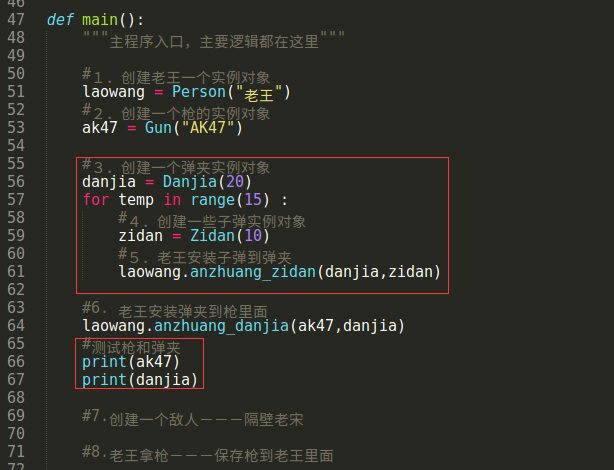
#9.老王开枪打敌人

if \_\_name\_\_ == '\_\_main\_\_':

main()

### 6.2.5. 测试枪和弹夹







class Person(object):

"""人的类"""

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

def anzhuang\_zidan(self,danjia\_temp,zidan\_temp):

"""安装子弹到弹夹"""

#弹夹.保存(子弹)

danjia\_temp.baocun\_zidan(zidan\_temp)

def anzhuang\_danjia(self,gun\_temp,danjia\_temp):

"""安装弹夹到枪里面"""

#枪.保存(弹夹)

gun\_temp.baocun\_danjia(danjia\_temp)

class Gun(object):

"""枪类"""

def \_\_init\_\_(self, name):

super(Gun, self).\_\_init\_\_()

self.name = name

self.danjia = None#默认是没有弹夹

def baocun\_danjia(self,danjia\_temp):

"""保存弹夹到枪里面"""

self.danjia = danjia\_temp

def \_\_str\_\_(self):

return "%s里有%s"%(self.name,self.danjia)

class Danjia(object):

"""弹夹类"""

def \_\_init\_\_(self, max\_num):

super(Danjia, self).\_\_init\_\_()

self.max\_num = max\_num#最多装20发子弹

self.zidan\_list = []#装多颗子弹

def baocun\_zidan(self,zidan\_temp):

"""保存子弹到弹夹里面"""

self.zidan\_list.append(zidan\_temp)

def \_\_str\_\_(self):

return "弹夹信息:%d/%d"%(len(self.zidan\_list),self.max\_num)

class Zidan(object):

"""子弹类"""

def \_\_init\_\_(self, sha\_shang\_li):

super(Zidan, self).\_\_init\_\_()

self.sha\_shang\_li = sha\_shang\_li

def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

laowang = Person("老王")

#２．创建一个枪的实例对象

ak47 = Gun("AK47")

#３．创建一个弹夹实例对象

danjia = Danjia(20)

for temp in range(15) :

#４．创建一些子弹实例对象

zidan = Zidan(10)

#５．老王安装子弹到弹夹

laowang.anzhuang\_zidan(danjia,zidan)

#6.　老王安装弹夹到枪里面

laowang.anzhuang\_danjia(ak47,danjia)

#测试枪和弹夹

print(ak47)

print(danjia)

#7.创建一个敌人－－－隔壁老宋

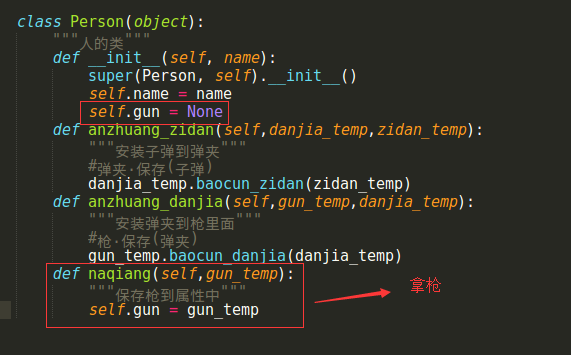
#8.老王拿枪－－－保存枪到老王里面

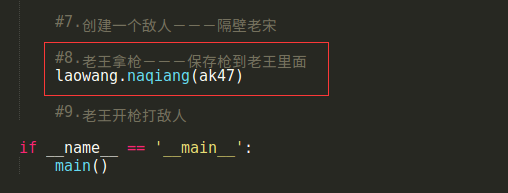
#9.老王开枪打敌人

if \_\_name\_\_ == '\_\_main\_\_':

main()

### 6.2.6. 老王拿枪





class Person(object):

"""人的类"""

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

self.gun = None

def anzhuang\_zidan(self,danjia\_temp,zidan\_temp):

"""安装子弹到弹夹"""

#弹夹.保存(子弹)

danjia\_temp.baocun\_zidan(zidan\_temp)

def anzhuang\_danjia(self,gun\_temp,danjia\_temp):

"""安装弹夹到枪里面"""

#枪.保存(弹夹)

gun\_temp.baocun\_danjia(danjia\_temp)

def naqiang(self,gun\_temp):

"""保存枪到属性中"""

self.gun = gun\_temp

class Gun(object):

"""枪类"""

def \_\_init\_\_(self, name):

super(Gun, self).\_\_init\_\_()

self.name = name

self.danjia = None#默认是没有弹夹

def baocun\_danjia(self,danjia\_temp):

"""保存弹夹到枪里面"""

self.danjia = danjia\_temp

def \_\_str\_\_(self):

return "%s里有%s"%(self.name,self.danjia)

class Danjia(object):

"""弹夹类"""

def \_\_init\_\_(self, max\_num):

super(Danjia, self).\_\_init\_\_()

self.max\_num = max\_num#最多装20发子弹

self.zidan\_list = []#装多颗子弹

def baocun\_zidan(self,zidan\_temp):

"""保存子弹到弹夹里面"""

self.zidan\_list.append(zidan\_temp)

def \_\_str\_\_(self):

return "弹夹信息:%d/%d"%(len(self.zidan\_list),self.max\_num)

class Zidan(object):

"""子弹类"""

def \_\_init\_\_(self, sha\_shang\_li):

super(Zidan, self).\_\_init\_\_()

self.sha\_shang\_li = sha\_shang\_li

def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

laowang = Person("老王")

#２．创建一个枪的实例对象

ak47 = Gun("AK47")

#３．创建一个弹夹实例对象

danjia = Danjia(20)

for temp in range(15) :

#４．创建一些子弹实例对象

zidan = Zidan(10)

#５．老王安装子弹到弹夹

laowang.anzhuang\_zidan(danjia,zidan)

#6.　老王安装弹夹到枪里面

laowang.anzhuang\_danjia(ak47,danjia)

#测试枪和弹夹

#print(ak47)

#print(danjia)

#7.创建一个敌人－－－隔壁老宋

#8.老王拿枪－－－保存枪到老王里面

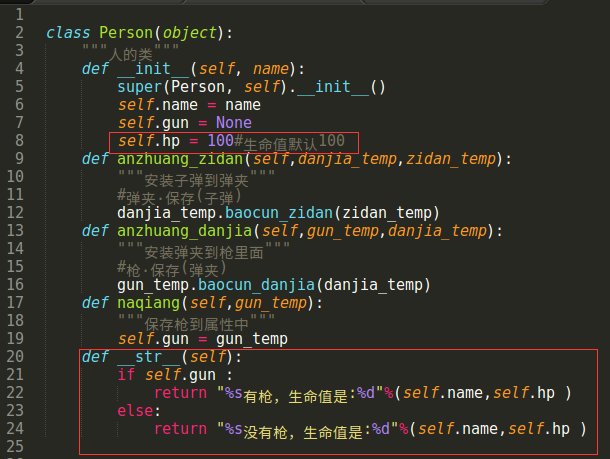
laowang.naqiang(ak47)

#9.老王开枪打敌人

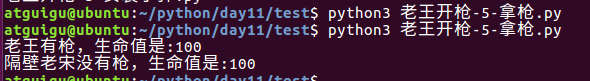
if \_\_name\_\_ == '\_\_main\_\_':

main()

### 6.2.7. 创建一个敌人







class Person(object):

"""人的类"""

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

self.gun = None

self.hp = 100#生命值默认100

def anzhuang\_zidan(self,danjia\_temp,zidan\_temp):

"""安装子弹到弹夹"""

#弹夹.保存(子弹)

danjia\_temp.baocun\_zidan(zidan\_temp)

def anzhuang\_danjia(self,gun\_temp,danjia\_temp):

"""安装弹夹到枪里面"""

#枪.保存(弹夹)

gun\_temp.baocun\_danjia(danjia\_temp)

def naqiang(self,gun\_temp):

"""保存枪到属性中"""

self.gun = gun\_temp

def \_\_str\_\_(self):

if self.gun :

return "%s有枪，生命值是:%d"%(self.name,self.hp )

else:

return "%s没有枪，生命值是:%d"%(self.name,self.hp )

class Gun(object):

"""枪类"""

def \_\_init\_\_(self, name):

super(Gun, self).\_\_init\_\_()

self.name = name

self.danjia = None#默认是没有弹夹

def baocun\_danjia(self,danjia\_temp):

"""保存弹夹到枪里面"""

self.danjia = danjia\_temp

def \_\_str\_\_(self):

return "%s里有%s"%(self.name,self.danjia)

class Danjia(object):

"""弹夹类"""

def \_\_init\_\_(self, max\_num):

super(Danjia, self).\_\_init\_\_()

self.max\_num = max\_num#最多装20发子弹

self.zidan\_list = []#装多颗子弹

def baocun\_zidan(self,zidan\_temp):

"""保存子弹到弹夹里面"""

self.zidan\_list.append(zidan\_temp)

def \_\_str\_\_(self):

return "弹夹信息:%d/%d"%(len(self.zidan\_list),self.max\_num)

class Zidan(object):

"""子弹类"""

def \_\_init\_\_(self, sha\_shang\_li):

super(Zidan, self).\_\_init\_\_()

self.sha\_shang\_li = sha\_shang\_li

def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

laowang = Person("老王")

#２．创建一个枪的实例对象

ak47 = Gun("AK47")

#３．创建一个弹夹实例对象

danjia = Danjia(20)

for temp in range(15) :

#４．创建一些子弹实例对象

zidan = Zidan(10)

#５．老王安装子弹到弹夹

laowang.anzhuang\_zidan(danjia,zidan)

#6.　老王安装弹夹到枪里面

laowang.anzhuang\_danjia(ak47,danjia)

#测试枪和弹夹

#print(ak47)

#print(danjia)

#7.创建一个敌人－－－隔壁老宋

gebi\_laosong = Person("隔壁老宋")

#8.老王拿枪－－－保存枪到老王里面

laowang.naqiang(ak47)

print(laowang)

print(gebi\_laosong)

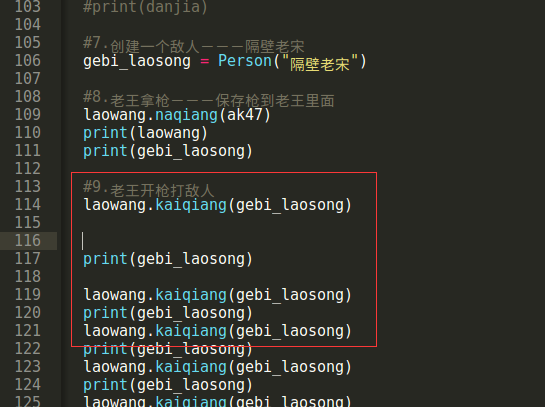
#9.老王开枪打敌人

if \_\_name\_\_ == '\_\_main\_\_':

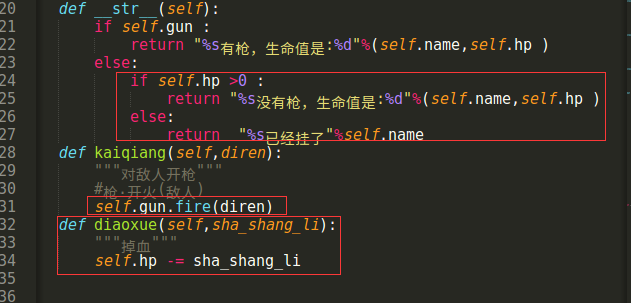
main()

### 6.2.8. 老王开枪打敌人

#### 主程序中



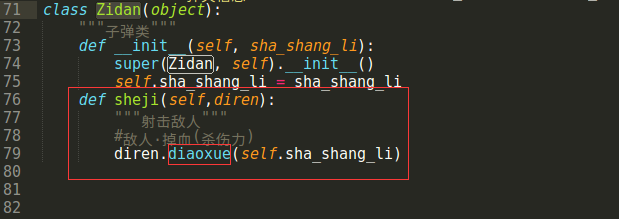
#### 人类类里面



#### 枪类里面



#### 子弹类



### 6.2.9. 完整代码

class Person(object):

"""人的类"""

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

self.gun = None

self.hp = 100#生命值默认100

def anzhuang\_zidan(self,danjia\_temp,zidan\_temp):

"""安装子弹到弹夹"""

#弹夹.保存(子弹)

danjia\_temp.baocun\_zidan(zidan\_temp)

def anzhuang\_danjia(self,gun\_temp,danjia\_temp):

"""安装弹夹到枪里面"""

#枪.保存(弹夹)

gun\_temp.baocun\_danjia(danjia\_temp)

def naqiang(self,gun\_temp):

"""保存枪到属性中"""

self.gun = gun\_temp

def \_\_str\_\_(self):

if self.gun :

return "%s有枪，生命值是:%d"%(self.name,self.hp )

else:

if self.hp >0 :

return "%s没有枪，生命值是:%d"%(self.name,self.hp )

else:

return "%s已经挂了"%self.name

def kaiqiang(self,diren):

"""对敌人开枪"""

#枪.开火(敌人)

self.gun.fire(diren)

def diaoxue(self,sha\_shang\_li):

"""掉血"""

self.hp -= sha\_shang\_li

class Gun(object):

"""枪类"""

def \_\_init\_\_(self, name):

super(Gun, self).\_\_init\_\_()

self.name = name

self.danjia = None#默认是没有弹夹

def baocun\_danjia(self,danjia\_temp):

"""保存弹夹到枪里面"""

self.danjia = danjia\_temp

def \_\_str\_\_(self):

return "%s里有%s"%(self.name,self.danjia)

def fire(self,diren):

"""开火"""

#从弹夹里得到一颗子弹

zidan\_temp = self.danjia.tanchu\_zidan()

#子弹.射击(敌人)

zidan\_temp.sheji(diren)

class Danjia(object):

"""弹夹类"""

def \_\_init\_\_(self, max\_num):

super(Danjia, self).\_\_init\_\_()

self.max\_num = max\_num#最多装20发子弹

self.zidan\_list = []#装多颗子弹

def baocun\_zidan(self,zidan\_temp):

"""保存子弹到弹夹里面"""

self.zidan\_list.append(zidan\_temp)

def tanchu\_zidan(self):

"""弹夹中得到一个末尾的子弹"""

if self.zidan\_list :

return self.zidan\_list.pop()

else:

return None

def \_\_str\_\_(self):

return "弹夹信息:%d/%d"%(len(self.zidan\_list),self.max\_num)

class Zidan(object):

"""子弹类"""

def \_\_init\_\_(self, sha\_shang\_li):

super(Zidan, self).\_\_init\_\_()

self.sha\_shang\_li = sha\_shang\_li

def sheji(self,diren):

"""射击敌人"""

#敌人.掉血(杀伤力)

diren.diaoxue(self.sha\_shang\_li)

def main():

"""主程序入口，主要逻辑都在这里"""

#１．创建老王一个实例对象

laowang = Person("老王")

#２．创建一个枪的实例对象

ak47 = Gun("AK47")

#３．创建一个弹夹实例对象

danjia = Danjia(20)

for temp in range(15) :

#４．创建一些子弹实例对象

zidan = Zidan(10)

#５．老王安装子弹到弹夹

laowang.anzhuang\_zidan(danjia,zidan)

#6.　老王安装弹夹到枪里面

laowang.anzhuang\_danjia(ak47,danjia)

#测试枪和弹夹

#print(ak47)

#print(danjia)

#7.创建一个敌人－－－隔壁老宋

gebi\_laosong = Person("隔壁老宋")

#8.老王拿枪－－－保存枪到老王里面

laowang.naqiang(ak47)

print(laowang)

print(gebi\_laosong)

#9.老王开枪打敌人

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

laowang.kaiqiang(gebi\_laosong)

print(gebi\_laosong)

if \_\_name\_\_ == '\_\_main\_\_':

main()