OOP七大设计原则-隔离接口原则

概述：
    应该使用多个隔离的接口或抽象类，而不是用单个的总接口或抽象类．应尽量细化接口，接口中的方法尽量少，依赖几个隔离的接口要比依赖于庞大的综合接口要灵活得多．接口本身就是软件设计时对外部提供的规范和契约，通过多个接口分散定义多个单一的接口，可以降低耦合、预防需求变更带来的代码修改和故障几率，提高系统的灵活性、可扩展性和维护成本．Java的接口及Python的多重继承都是支撑接口隔离原则的基础特性．(注：JAVA中的子类不能继承多个父类)．

适用场景：
    对臃肿的综合接口拆分，原系统定义了一个客户管理的接口，该接口中有多个方法，有供公司内部调用的，有供第三方调用的，还有供Mobile调用，从单一职责的角度来看好像也是合理的，因为它只提供客户管理的接口功能，但从接口隔离原则来看，接口中的模块数量及功能都过多，耦合度太高，需要细化将总接口拆分成多个专用的单一接口．

案例场景：
    以学校的十佳评选为例，要求入选的同学必须"德智体美劳"全面领先．

未遵守隔离接口原则的代码如下：

In [1]:
```
from abc import ABCMeta, abstractmethod

class best_student(object):
    def __init__(self, name):
        self.name = name

    @abstractmethod
    def de(self):
        pass

    @abstractmethod
    def zhi(self):
        pass

    @abstractmethod
    def ti(self):
        pass

    @abstractmethod
    def mei(self):
        pass

    @abstractmethod
    def lao(self):
        pass

class TopTenStudent(best_student):
    def de(self):
        print("%s 品德好!" %self.name)

    def zhi(self):
        print('%s 智商高!' %self.name)

    def ti(self):
        print('%s 身体好!' %self.name)

    def mei(self):
        print('%s 懂审美!' %self.name)

    def lao(self):
        print('%s 爱劳动!' %self.name)


class Judge(object):
    __metaclass__ = ABCMeta
    def __init__(self, best_student):
        self.best_student = best_student
    @abstractmethod
    def show(self):
        pass

class schoolJudge(Judge):
    def show(self):
        self.best_student.de()
        self.best_student.zhi()
        self.best_student.ti()
        self.best_student.mei()
        self.best_student.lao()

if __name__ == '__main__':
    s = TopTenStudent("lily")
    schoolJudge(s).show()
```

```
lily 品德好!
lily 智商高!
lily 身体好!
lily 懂审美!
lily 爱劳动!
```

    现在新的需求来了，要求选拔一批优秀学生参加奥数比赛，另一批参加体育比赛．已有的选拔标准太全面，显然不符合新需求，所以我们需要拆分原有的优秀学生评选接口，根据新需求可以先把"智商高"和"体育棒"拆分出来单独各自组成新的评选接口．

示例代码：

In [1]:
```
#coding:utf8

from abc import ABCMeta, abstractmethod

class best_zhi(object):
    __metaclass__ = ABCMeta
    @abstractmethod
    def zhi(self):
        pass

class best_ti(object):
    __metaclass__ = ABCMeta

    @abstractmethod
    def ti(self):
        pass

class best_student(object):

    @abstractmethod
    def de(self):
        pass

    @abstractmethod
    def mei(self):
        pass

    @abstractmethod
    def lao(self):
        pass


class TopZhiStuent(best_zhi):

    def __init__(self, name):
        self.name = name

    def zhi(self):
        print("%s 智商高!" %self.name)


class TopTiStudent(best_ti):

    def __init__(self, name):
        self.name  = name

    def ti(self):
        print('%s 体育好' %self.name)


class TopTenStudent(best_student, TopZhiStuent, TopTiStudent):

    def __init__(self, name):
        self.name = name

    def de(self):
        print("%s 品德好!" %self.name)

    def mei(self):
        print('%s 懂审美!' %self.name)

    def lao(self):
        print('%s 爱劳动!' %self.name)


class Judge(object):
    __metaclass__ = ABCMeta
    def __init__(self, best_student):
        self.best_student = best_student
    @abstractmethod
    def show(self):
        pass

class TopZhiJudge(Judge):
    def show(self):
        self.best_student.zhi()

class TopTiJudge(Judge):
    def show(self):
        self.best_student.ti()

class TopTenJudge(Judge):
    def show(self):
        self.best_student.zhi()
        self.best_student.de()
        self.best_student.ti()
        self.best_student.mei()
        self.best_student.lao()

if __name__ == '__main__':
    s = TopZhiStuent("lily")
    TopZhiJudge(s).show()
    print('##' * 10)
    s = TopTiStudent("Tom")
    TopTiJudge(s).show()
    print('##' * 10)
    s = TopTenStudent("lucy")
    TopTenJudge(s).show()
```

```
lily 智商高!
####################
Tom 体育好
####################
lucy 智商高!
lucy 品德好!
lucy 体育好
lucy 懂审美!
lucy 爱劳动!
```

注意事项：
    1)为客户端提供尽可能小的单独的接口，而不是提供大的总接口.
    2)注意控制接口的粒度，接口太小会导致系统中接口数量的爆炸，不利于维护；接口太大将违背接口隔离原则，灵活性较差.
    3)客户端不应该依赖于自身不需要的接口，一个类对另一个类的依赖应该建立在最小接口上.