单例模式

实例

单例模式确保一个类只有一个实例，并提供一个全局访问点。

Python 实现单例模式的示例：

```
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(Singleton, cls).__new__(cls, *args, **kwargs)
        return cls._instance

# 使用单例模式
singleton1 = Singleton()
singleton2 = Singleton()

print(singleton1 is singleton2)  # 输出: True
```

特点

- **唯一性**：确保一个类只有一个实例。
- **全局访问点**：提供一个访问该实例的全局访问点。
- **延迟实例化**：可以延迟实例化，只有在需要时才创建实例。

建造者模式

实例

建造者模式将一个复杂对象的构建过程与其表示分离，使得同样的构建过程可以创建不同的表示。

Python 实现建造者模式的示例：

```
class Product:
    def __init__(self):
        self.parts = []

    def add(self, part):
        self.parts.append(part)

    def show(self):
        print("Product parts:", self.parts)

class Builder:
    def build_part_a(self):
        pass
```

```
    def build_part_b(self):
        pass

    def get_result(self):
        pass

class ConcreteBuilder(Builder):
    def __init__(self):
        self.product = Product()

    def build_part_a(self):
        self.product.add("Part A")

    def build_part_b(self):
        self.product.add("Part B")

    def get_result(self):
        return self.product

class Director:
    def __init__(self, builder):
        self.builder = builder

    def construct(self):
        self.builder.build_part_a()
        self.builder.build_part_b()

# 使用建造者模式
builder = ConcreteBuilder()
director = Director(builder)
director.construct()
product = builder.get_result()
product.show()  # 输出: Product parts: ['Part A', 'Part B']
```

特点

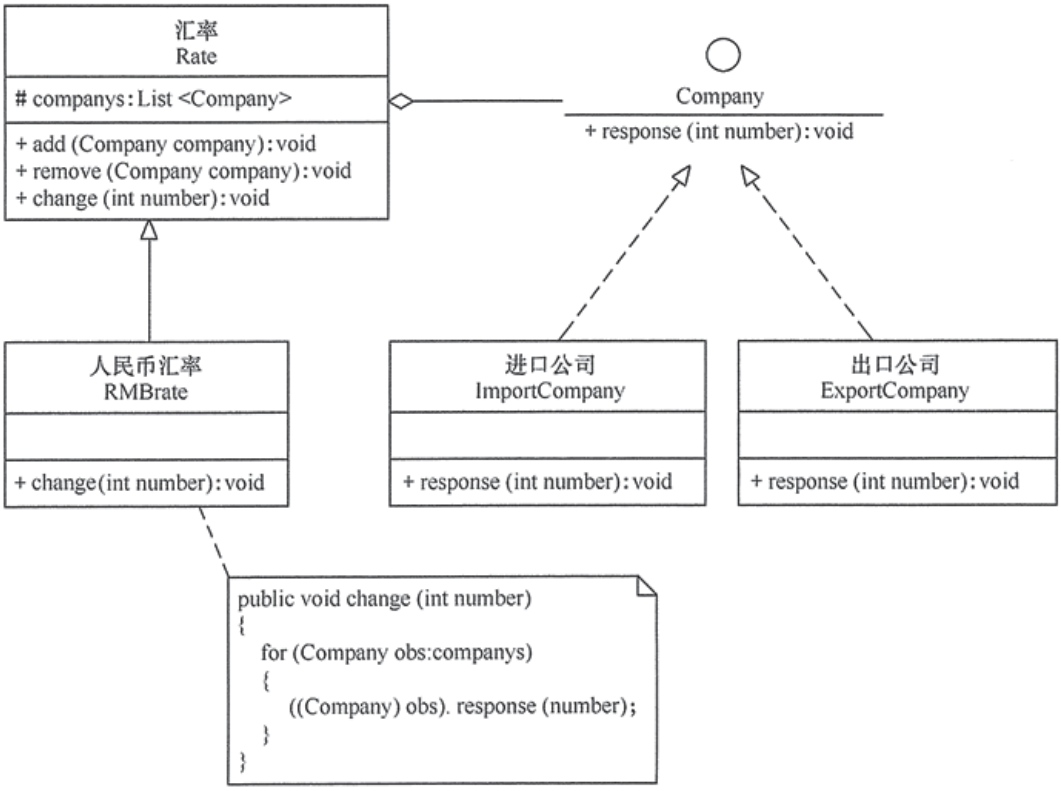
- **复杂对象创建**：适用于创建复杂对象。
- **分步骤构建**：将对象的创建步骤分开，可以逐步构建对象。
- **独立于表示**：将构建过程与表示分离，同样的构建过程可以创建不同的表示。
- **可扩展性**：通过不同的建造者实现，可以创建不同类型的对象。

- 观察者模式：观察者（Observer）是指多个对象间存在一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。这种模式有时又称作发布-订阅模式、模型-视图模式，它是对象行为型模式。
  - 模式动机：建立一种对象与对象之间的依赖关系，一个对象发生改变时将自动通知其他对象，其他对象将相应做出反应。在此，发生改变的对象称为观察目标，而被通知的对象称为观察者，一个观察目标可以对应多个观察者，而且这些观察者之间没有相互联系，可以根据需要增加和删除观察者，使得系统更易于扩展，这就是观察者模式的模式动机。
  - 实例：

    利用观察者模式设计一个程序，分析人民币汇率的升值或贬值对进口公司进口产品成本或出口公司的出口产品收入以及公司利润率的影响。

    分析：当“人民币汇率”升值时，进口公司的进口产品成本降低且利润率提升，出口公司的出口产品收入降低且利润率降低；当“人民币汇率”贬值时，进口公司的进口产品成本提升且利润率降低，出口公司的出口产品收入提升且利润率提升。

    这里的汇率（Rate）类是抽象目标类，它包含了保存观察者（Company）的 List 和增加/删除观察者的方法，以及有关汇率改变的抽象方法 change(int number)；而人民币汇率（RMBrate）类是具体目标， 它实现了父类的 change(int number) 方法，即当人民币汇率发生改变时通过相关公司；公司（Company）类是抽象观察者，它定义了一个有关汇率反应的抽象方法 response(int number)；进口公司（ImportCompany）类和出口公司（ExportCompany）类是具体观察者类，它们实现了父类的 response(int number) 方法，即当它们接收到汇率发生改变的通知时作为相应的反应。下图所示是其 UML 结构图。



  - 观察者模式的优点：
    - 具体目标和具体观察者是松耦合关系，观察者接口的引入降低了系统的复杂度；
    - 观察模式满足“开-闭原则”。目标接口仅仅依赖于观察者接口，系统的可复用性和可扩展性得到提升。
  - 观察者模式的缺点：
    - 如果一个观察者关联多实例的话，将所有的实例都通知到会花费很多时间；
    - 如果在观察者和观察目标之间有循环依赖的话，观察目标会触发它们之间进行循环调用，可能导致系统崩溃；
    - 观察者模式没有相应的机制让观察者知道所观察的目标对象是怎么发生变化的，而仅仅只是知道观察目标发生了变化。

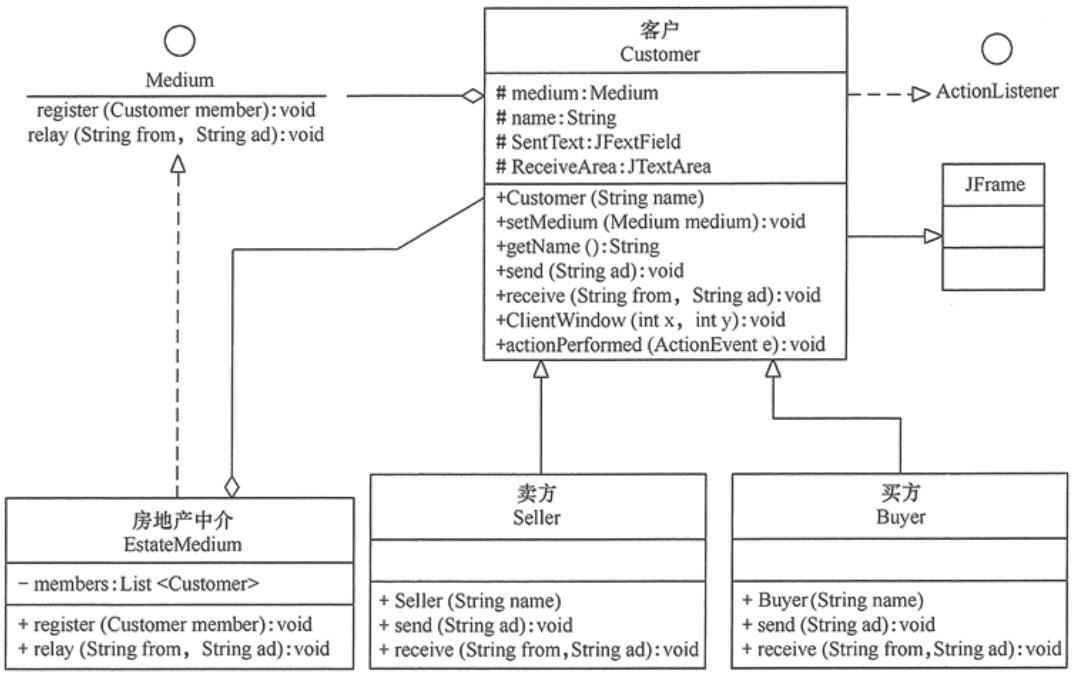
- 中介者模式：定义一个中介对象来封装一系列对象之间的交互，使原有对象之间的耦合松散，且可以独立地改变它们之间的交互。中介者模式又叫调停模式，它是迪米特法则的典型应用。
  - 实例：

    用中介者模式编写一个“房地产交流平台”程序。

    分析：首先，定义一个中介公司（Medium）接口，它是抽象中介者，它包含了客户注册方法 register(Customer member) 和信息转发方法 relay(String from,String ad)；再定义一个房地产中介（EstateMedium）公司，它是具体中介者类，它包含了保存客户信息的 List 对象，并实现了中介公司中的抽象方法。

    然后，定义一个客户（Customer）类，它是抽象同事类，其中包含了中介者的对象，和发送信息的 send(String ad) 方法与接收信息的 receive(String from，String ad) 方法的接口，由于本程序是窗体程序，所以本类继承 JPmme 类，并实现动作事件的处理方法 actionPerformed(ActionEvent e)。

    最后，定义卖方（Seller）类和买方（Buyer）类，它们是具体同事类，是客户（Customer）类的子类，它们实现了父类中的抽象方法，通过中介者类进行信息交流，其结构 UML 图如下：



  - 中介者模式的优点为：
    - 类之间各司其职，符合迪米特法则；
    - 降低了对象之间的耦合性，使得对象易于独立地被复用；
    - 将对象间的一对多关联转变为一对一的关联，提高系统的灵活性，使得系统易于维护和扩展；
  - 中介者模式的缺点为：
    - 中介者模式将原本多个对象直接的相互依赖变成了中介者和多个同事类的依赖关系。当同事类越多时，中介者就会越臃肿，变得复杂且难以维护。