**题目：**

小王正在为某公司设计开发一套物业租赁管理系统，该公司有多种类型的物业，如公寓、商铺等，并且在将来可能会增加新的物业类型，如别墅、车库等；公司的经纪每租出一个物业，主管就会收到相应的租赁信息。请你针对上面的描述，帮助小王选择合适的设计模式进行设计。

1)简要说明你的设计思路和选择的模式。

2)给出你设计的UML类图和实现代码。

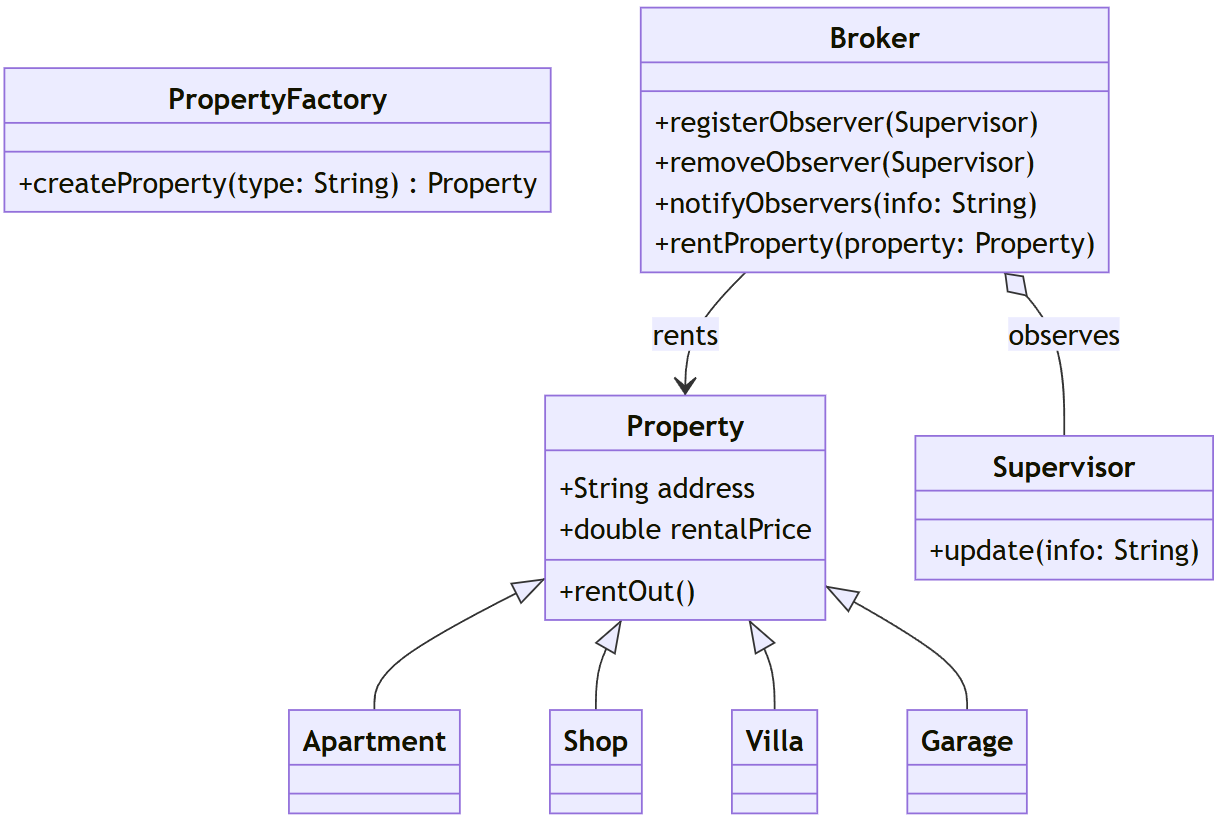
**设计思路与模式：**

可以使用**工厂方法模式和观察者模式。**

物业有多种类型（如公寓、商铺等），未来可能会新增类型（别墅、车库等）。工厂方法模式允许创建者通过特定的工厂方法来创建具体物业类型的实例，从而满足灵活扩展需求。

在物业租出后，需要自动通知主管。使用观察者模式可以在每次成功租出物业时通知主管，不同的物业类型可以在租赁时触发观察者的通知，实现信息的及时推送。

**类图：**



|  |
| --- |
| **代码** |
| import java.util.ArrayList;  import java.util.List;  *// 抽象物业类*  abstract class Property {      protected String address;      protected double rentalPrice;      public Property(String address, double rentalPrice) {          this.address = address;          this.rentalPrice = rentalPrice;      }      public abstract void rentOut();  }  *// 公寓类*  class Apartment extends Property {      public Apartment(String address, double rentalPrice) {          super(address, rentalPrice);      }      @Override      public void rentOut() {          System.out.println("Apartment rented out at " + address);      }  }  *// 商铺类*  class Shop extends Property {      public Shop(String address, double rentalPrice) {          super(address, rentalPrice);      }      @Override      public void rentOut() {          System.out.println("Shop rented out at " + address);      }  }  *// 工厂类，用于创建不同类型的物业*  class PropertyFactory {      public Property createProperty(String type, String address, double rentalPrice) {          switch (type) {              case "Apartment":                  return new Apartment(address, rentalPrice);              case "Shop":                  return new Shop(address, rentalPrice);  *// 其他物业类型可以在此处添加*              default:                  throw new IllegalArgumentException("Unknown property type.");          }      }  }  *// 观察者接口*  interface Observer {      void update(String info);  }  *// 主管类，接收租赁信息*  class Supervisor implements Observer {      private String name;      public Supervisor(String name) {          this.name = name;      }      @Override      public void update(String info) {          System.out.println(name + " received rental info: " + info);      }  }  *// 经纪人类，管理物业租赁并通知主管*  class Broker {      private List<Observer> supervisors = new ArrayList<>();      public void registerObserver(Observer supervisor) {          supervisors.add(supervisor);      }      public void removeObserver(Observer supervisor) {          supervisors.remove(supervisor);      }      public void notifyObservers(String info) {          for (Observer supervisor : supervisors) {              supervisor.update(info);          }      }      public void rentProperty(Property property) {          property.rentOut();          notifyObservers("Property rented out at " + property.address + " for $" + property.rentalPrice);      }  }  *// 测试类*  public class RentalManagementSimulation {      public static void main(String[] args) {  *// 创建工厂*          PropertyFactory factory = new PropertyFactory();  *// 创建物业*          Property apartment = factory.createProperty("Apartment", "123 Street", 1500);          Property shop = factory.createProperty("Shop", "456 Avenue", 3000);  *// 创建经纪人和主管*          Broker broker = new Broker();          Supervisor supervisor = new Supervisor("John");          broker.registerObserver(supervisor);  *// 经纪人租出物业*          broker.rentProperty(apartment);          broker.rentProperty(shop);      }  } |