微光招新

Java06

task1

1. 如图
2. 如图

```
}
class Dish_1 extends Dish_0{//不能加public，因为一个class文件只能有一个public类  2个用法
    public Dish_1(){  1个用法
        super( name: "麻婆豆腐", price: 11);//不能直接访问父类的name和price，因为是private的，所以我们调用父类有参构造器来初始化name和price
        //一般子类构造器被调用时会先调用父类无参构造器，我们要求子类构造器调用父类有参构造器，所以要加super（）
    }
    //因为要求profile方法要适合该菜品，所以要重写profile方法

    @Override//方法重写的标识，可以校验这个重写的方法的名字，参数，返回值类型是不是跟父类一致  0个用法
    public void profile(){
        System.out.println("麻婆豆腐是经典川菜，麻辣鲜香");
    }
}
//Dish_2类和Dish_1类同理
class Dish_2 extends Dish_0{  0个用法
    public Dish_2(){  0个用法
        super( name: "回锅肉",  price: 20);
    }


    @Override  0个用法
    public void profile(){
        System.out.println("回锅肉要炒2次，好吃");
    }
}
```

task2

3. 如图

```java
package com.LILY.objecttask06;

public interface Order {//接口类一般定义常量和抽象方法  3 个用法  2 个实现
    public abstract void cook();  0 个用法  2 个实现

    public abstract boolean check();  0 个用法  2 个实现
}

```

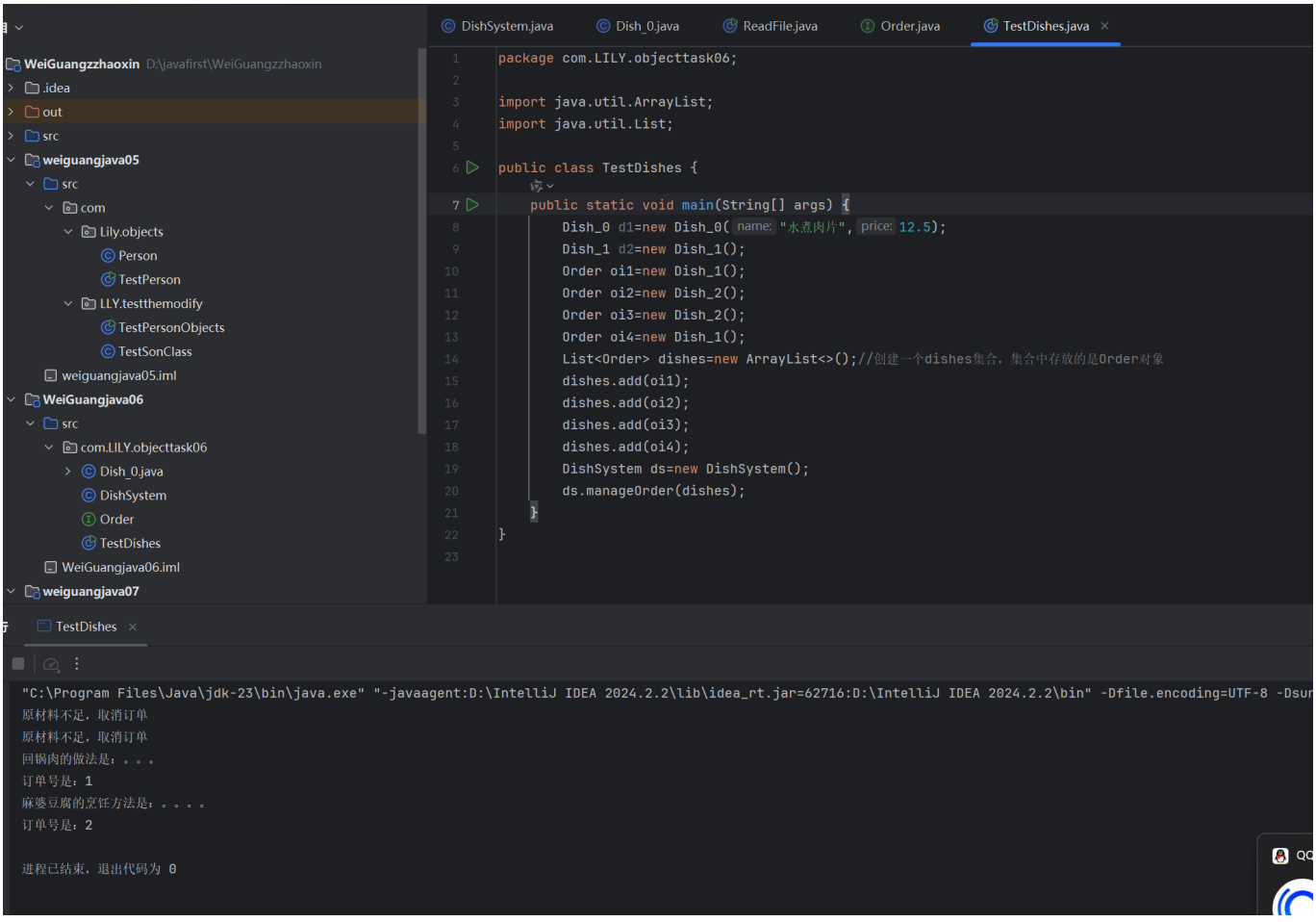
```java
public class Dish_0 {  4 个用法  2 个继承者

}
class Dish_1 extends Dish_0 implements Order{//不能加public，因为一个class文件只能有一个public类  3 个用法
    public Dish_1(){  2 个用法
        super( name: "麻婆豆腐", price: 11);//不能直接访问父类的name和price，因为是private的，所以我们调用父类有参构造器来初始化name和price
        //一般子类构造器被调用时会先调用父类无参构造器，我们要求子类构造器调用父类有参构造器，所以要加super（）
    }
    //因为要求profile方法要适合该菜品，所以要重写profile方法

    @Override//方法重写的标识，可以校验这个重写的方法的名字，参数，返回值类型是不是跟父类一致  0 个用法
    public void profile(){
        System.out.println("麻婆豆腐是经典川菜，麻辣鲜香");
    }

    @Override  1 个用法
    public boolean check(){//随机数决定是true还是false
        boolean a;
        Random r=new Random();
        int n=r.nextInt( bound: 10);//随机生成一个数用来判断check返回值
        if(n<5){
            a=false;
        }else{
            a = true;
        }
        return a;
    }

    @Override  1 个用法
    public void cook(){
        System.out.println("麻婆豆腐的烹饪方法是：。。。。");
    }
}
```

```java
//Dish_2类和Dish_1类同理
class Dish_2 extends Dish_0 implements Order{  1 个用法
    public Dish_2(){  1 个用法
        super( name: "回锅肉",  price: 20);
    }

    @Override  0 个用法
    public void profile(){
        System.out.println("回锅肉，好吃");
    }

    @Override  1 个用法
    public boolean check(){
        boolean a;
        double n=Math.random()*10;//随机生成一个数用来判断check返回值
        if(n<5.0){
            a = false;
        }else{
            a = true;
        }
        return a;
    }

    @Override  1 个用法
```

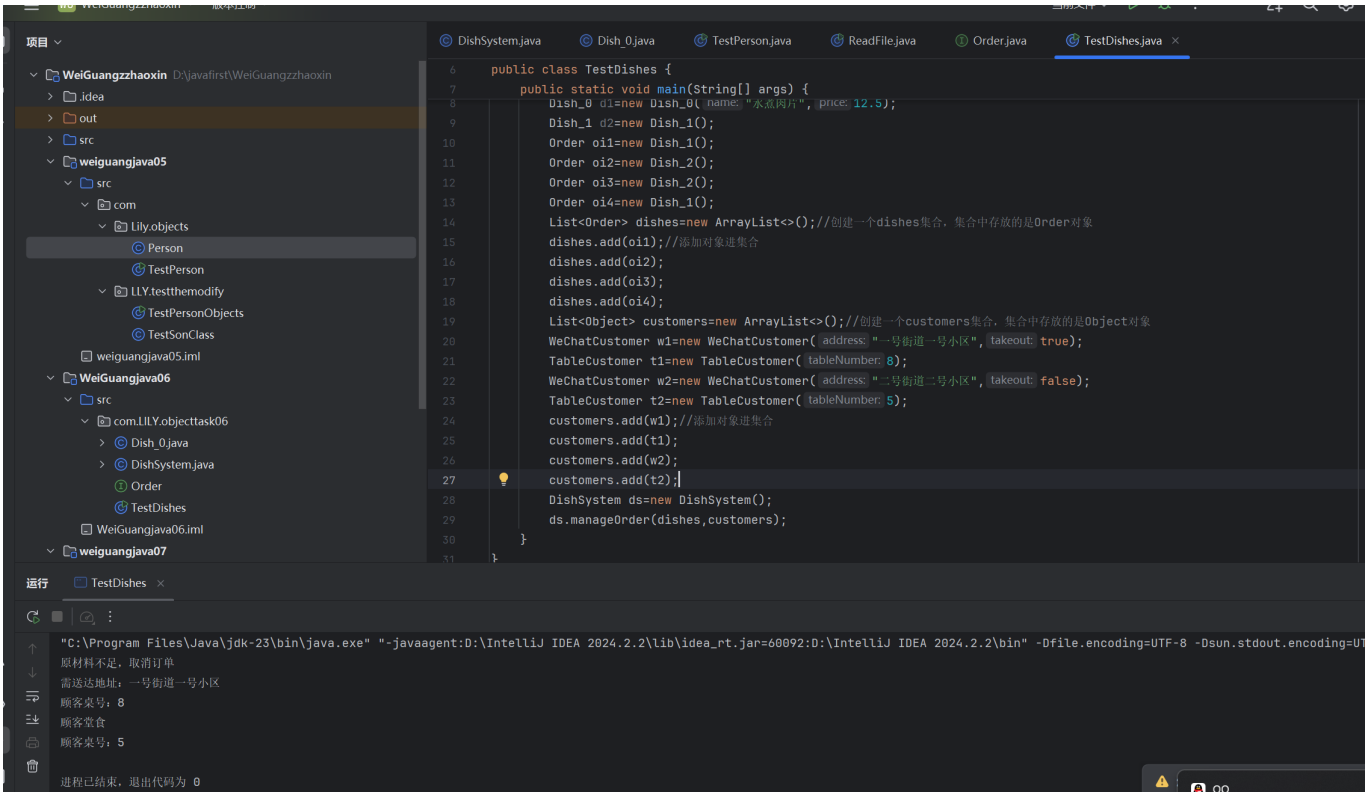
```
    public void cook() { System.out.println("回锅肉的做法是：。。。"); }
}
```

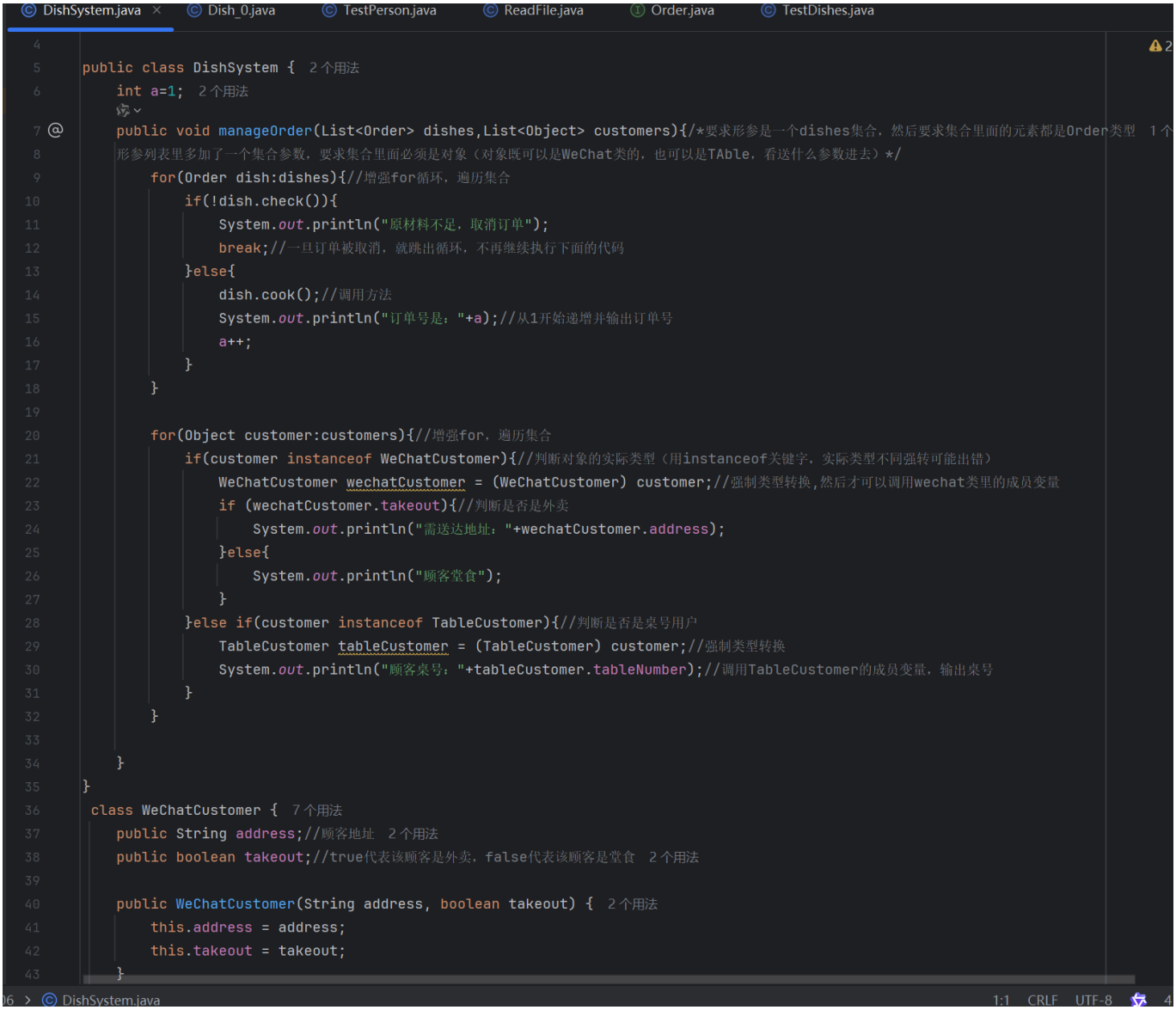
DishSystem.java | Dish_0.java | ReadFile.java | Order.java | TestDishes.java

```
package com.LILY.objecttask06;

import java.util.List;

public class DishSystem {  2 个用法
    int a=1;  2 个用法
    public void manageOrder(List<Order> dishes){//要求形参是一个dishes集合，然后要求集合里面的元素都是Order类型  1 个用法
        for(Order dish:dishes){//不知道索引数量（订单是要自己去调用Order定义对象再自己添加进集合里吗，我没太清楚这个），增强for
            if(!dish.check()){
                System.out.println("原材料不足，取消订单");
                break;//一旦订单被取消，就跳出循环，不再继续执行下面的代码
            }else{
                dish.cook();//调用方法
                System.out.println("订单号是："+a);//从1开始递增并输出订单号
                a++;
            }
        }

    }
}
```



task3


项目
WeiGuangzzhaoxin D:\javafirst\WeiGuangzzhaoxin
.idea
out
src
weiguangjava05
src
com
Lily.objects
Person
TestPerson
LLY.testthemodify
TestPersonObjects
TestSonClass
weiguangjava05.iml
WeiGuangjava06
src
com.LILY.objecttask06
Dish_0.java
DishSystem.java
Order
TestDishes
WeiGuangjava06.iml
weiguangjava07
DishSystem.java
Dish_0.java
TestPerson.java
ReadFile.java
Order.java
TestDishes.java
public class TestDishes {
public static void main(String[] args) {
Dish_0 d1=new Dish_0(name: "水煮肉片", price: 12.5);
Dish_1 d2=new Dish_1();
Order oi1=new Dish_1();
Order oi2=new Dish_2();
Order oi3=new Dish_2();
Order oi4=new Dish_1();
List<Order> dishes=new ArrayList<>();//创建一个dishes集合，集合中存放的是Order对象
dishes.add(oi1);//添加对象进集合
dishes.add(oi2);
dishes.add(oi3);
dishes.add(oi4);
List<Object> customers=new ArrayList<>();//创建一个customers集合，集合中存放的是Object对象
WeChatCustomer w1=new WeChatCustomer(address: "一号街道一号小区", takeout: true);
TableCustomer t1=new TableCustomer(tableNumber: 8);
WeChatCustomer w2=new WeChatCustomer(address: "二号街道二号小区", takeout: false);
TableCustomer t2=new TableCustomer(tableNumber: 5);
customers.add(w1);//添加对象进集合
customers.add(t1);
customers.add(w2);
customers.add(t2);
DishSystem ds=new DishSystem();
ds.manageOrder(dishes,customers);
}
}
运行
TestDishes
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D:\IntelliJ IDEA 2024.2.2\lib\idea_rt.jar=60092:D:\IntelliJ IDEA 2024.2.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UT
原材料不足，取消订单
需送达地址：一号街道一号小区
顾客桌号：8
顾客堂食
顾客桌号：5
进程已结束，退出代码为 0


DishSystem.java
Dish_0.java
TestPerson.java
ReadFile.java
Order.java
TestDishes.java
public class DishSystem { 2 个用法
int a=1; 2 个用法
public void manageOrder(List<Order> dishes,List<Object> customers){/*要求形参是一个dishes集合，然后要求集合里面的元素都是Order类型 1 个
形参列表里多加了一个集合参数，要求集合里面必须是对象（对象既可以是WeChat类的，也可以是TAble，看送什么参数进去）*/
for(Order dish:dishes){//增强for循环，遍历集合
if(!dish.check()){
System.out.println("原材料不足，取消订单");
break;//一旦订单被取消，就跳出循环，不再继续执行下面的代码
}else{
dish.cook();//调用方法
System.out.println("订单号是："+a);//从1开始递增并输出订单号
a++;
}
}
for(Object customer:customers){//增强for，遍历集合
if(customer instanceof WeChatCustomer){//判断对象的实际类型（用instanceof关键字，实际类型不同强转可能出错）
WeChatCustomer wechatCustomer = (WeChatCustomer) customer;//强制类型转换,然后才可以调用wechat类里的成员变量
if (wechatCustomer.takeout){//判断是否是外卖
System.out.println("需送达地址："+wechatCustomer.address);
}else{
System.out.println("顾客堂食");
}
}else if(customer instanceof TableCustomer){//判断是否是桌号用户
TableCustomer tableCustomer = (TableCustomer) customer;//强制类型转换
System.out.println("顾客桌号："+tableCustomer.tableNumber);//调用TableCustomer的成员变量，输出桌号
}
}
}
}
class WeChatCustomer { 7 个用法
public String address;//顾客地址 2 个用法
public boolean takeout;//true代表该顾客是外卖，false代表该顾客是堂食 2 个用法
public WeChatCustomer(String address, boolean takeout) { 2 个用法
this.address = address;
this.takeout = takeout;
}
DishSystem.java
1:1 CRLF UTF-8

```
    }
}

class TableCustomer {  3 个用法
    public int tableNumber;//顾客的桌号  2 个用法

    public TableCustomer(int tableNumber) {  0 个用法
        this.tableNumber = tableNumber;
    }
}
```