Task1.继承

1.继承代码编写

```
public class Dish {
    private String name;
    private double price;
    Dish (String name,double price){
        this.name=name;
        this.price=price;
    }
    public void profile() {}
    }
    class Dish_1 extends Dish{
    public Dish_1(){
        super("烤鸡",40);
    }
    @Override
    public  void profile(){
        System.out.println("烤鸡（Grilled chicken），是一道中餐菜肴"+"。主要原料有鸡肉、盐、味精、葱姜蒜、五香八角等"+"。制作者可依据自己的口味添加不同的调料"+"制作各种口味的烤鸡。但烤鸡较为油腻，食用宜适量。");
    }
    }
class Dish_2 extends Dish{
    public Dish_2(){
        super("火锅",30);
    }
    @Override
    public  void profile(){
        System.out.println("火锅，古称“古董羹”。" +"因食物投入沸水时发出的“咕咚”声而得名，是中国独创的美食之一，是"+"一种老少皆宜的食物烹饪方式。"+"火锅不仅是一种烹饪方式，也是一种文化的象征。");
    }
}
```

子类的profile通过方法重写可以覆盖父类的profile，于是就先调用了子类的方法。

Task2.接口和多态

查了资料发现要用到随机数来模拟true，false,需要导入随机数包，还需要用到list来遍历的对象（这里刚开始还不懂于是又花时间去搞明白了），而且也要导入个list，**但是编写过程中出现了System.out.print()与class System的冲突**,我只好把class System后加个1

```
import java.util.Random;
import java.util.List;
```

```
//导入随机数
/*创建接口*/
interface Order {
    void cook();
    boolean check();
}
/*创建两个类*/
class Dish_1 implements Order
{private static final Random ran = new Random();/*创建随机数对象用来cheak方法的使用
*/
    @Override
    public void cook() {/*重写cook用作修改*/
        System.out.print("烹饪方法：对半切");
    }
    @Override
    public boolean check() {
        return ran.nextBoolean();
    }}
class Dish_2 implements Order
{private static final Random ran = new Random();
    @Override
    public void cook() {
        System.out.print("烹饪方法：直接炸");
    }
    @Override
    public boolean check() {
        return ran.nextBoolean();
    }}
 public class System1 {
    private static int cnt=1;/*进行食谱的追踪*/
     boolean sigh=true;
    public void manageOrder(List<Order> dishes){
        for(Order dish:dishes)/*进行遍历*/
            if (!dish.check()) {
                sigh= false;
                System.out.println("取消订单");
                break; /*检查食材是否可用*/
            }
        if(sigh)
        for(Order dish:dishes)
        { dish.cook();
            System.out.print(" ");
            System.out.print(cnt);
            cnt+=1;/*输出单号*/
            System.out.println();
    }
    }
}
```

里面for循环的写法对我来说十分新颖，因为我之前只会写for(int i=1;i<=n;i++)这种形式，通过接口使得维护更方便了，不过代码还是有一些瑕疵

Task3.泛型

接着查资料，也用了一些函数

```
import java.util.Random;
import java.util.List;
//导入随机数
/*创建接口*/
interface Order {
    void cook();
    boolean check();
}
/*创建两个类*/
class Dish_1 implements Order
{private static final Random ran = new Random();/*创建随机数对象用来cheak方法的使用
*/
    @Override
    public void cook() {/*重写cook用作修改*/
        System.out.print("烹饪方法：对半切");
    }
    @Override
    public boolean check() {
        return ran.nextBoolean();
    }}
class Dish_2 implements Order
{private static final Random ran = new Random();
    @Override
    public void cook() {
        System.out.print("烹饪方法：直接炸");
    }
    @Override
    public boolean check() {
        return ran.nextBoolean();
    }}
/*新建一个接口*/
interface  Customer{
}
/*试着把两个顾客类型放进同一个接口，便于managerOrder函数的传入*/
class TableCustomer implements Customer {
    public int tableId;//餐桌编号
}
class WechatCustomer implements Customer{
    public String address;//顾客地址
    public boolean takeout;//true代表该顾客是外卖，false代表该顾客是堂食
}
 public class System1 {
    private static int cnt=1;/*进行食谱的追踪*/
     boolean sigh=true;
    public <T extends Customer>void manageOrder(List<Order> dishes,T cust){
        for(Order dish:dishes)/*进行遍历*/
            if (!dish.check()) {
```

```
                sigh= false;
                System.out.println("取消订单");
                break; /*检查食材是否可用*/
            }
        if(sigh)
        for(Order dish:dishes)
        { dish.cook();
            System.out.print(" ");
            System.out.print(cnt);
            cnt+=1;/*输出单号*/
            System.out.println();

        if(cust instanceof TableCustomer)/*使用instanceof检查顾客的类型*/ {
            TableCustomer tcust =(TableCustomer) cust;
            System.out.println("送到"+tcust.tableId+"桌");
        }
        else {WechatCustomer wcust=(WechatCustomer) cust;
            if(wcust.takeout){System.out.println("送到"+wcust.address);
            }
            else System.out.println("堂食");
        }}

    }
}
```

问题是我不知道订单和顾客是不是一一对应的，姑且算是一个对应一个好了

总结

继承，接口，泛型理解起来容易一点，但是写起来我就脑袋空空了，一方面要找写代码需要的函数，另一方面访问修饰符也考验了我一大把，最严重的是语法格式了，用idea一直编译查看自己哪写得不对，不好。idea的tab补全很好用，打个大写开头就帮我补完内容了，总之，菜就多练😂