软件工程Lab5

石睿欣 胡宵宵 宋怡景 张逸涵

一、根据需求，对复杂软件工程问题进行推理、分析，详尽阐述针对复杂软件工程问题而开展的总体设计和详细设计

本次lab增加了定制化开发的需求，要求针对不同系统加入一些定制因素。对于不同的定制化开发需求，负责架构的同学进行了详细的分析和设计，如下：

1. **语言切换、货币切换**

   1-1 推理和分析

   语言切换和货币切换都是对软件国际化的需求。这种就要求开发人员在软件设计和文档开发过程中，使得功能和代码设计能够处理多种语言和货币，同时在创建不同语言版本时，不需要重新设计源代码。我们小组成员讨论后认为：应使得我们的设计中语言和货币以“接口”性质存在，不需要知道某个字符串究竟是什么语言的，而是通过某一“接口”得到某种语言的字符串。这样设计，在修改语言版本的时候，我们只需改动接口连接的语言库，不需要改动源码。

   1-2 总体设计

   我们使用java的`ResourceBundle`技术来实现定制化开发。配置文件中存储当前系统的国别语言信息和货币信息，在系统启动时，由初始化程序读入，存到环境上下文对象中去。系统运行时，想要访问语言信息，就访问环境上下文对象获取相应的信息。配置文件就是可插拔的组件，想要切换语言或货币，只需切换成相应的配置文件，重启系统即可，并不影响源码。

   1-3 详细设计

   配置文件就是资源属性文件（`properties`），该文件中存储的是一个个键值对。语言方面，配置文件中设计时key值与之前lab中的`InfoConstant`和`FileConstatnt`保持一致，value则对应不同的语言。如在英文配置文件中`PASSWORD_INVALID = Invalid passwor!`,而在中文配置文件中`PASSWORD_INVALID = 密码无效`。同时考虑到数据库中可能存有一些商品信息，如红茶的name是redTea，可能会造成不同语言的需求，这个也应该存在配置文件里。只是lab5并未要求前端，还未写入配置文件。

   而货币方面，我们统一用 `CURRENCY` 这个键表示，键值则用一个字符串表示。该字符串需要能表示多种货币，每种货币必需的信息。我们对于这个字符串的设计是：以分号分割不同货币，以下划线*分割同一个货币的不同信息。如“HK$*1.25_HDK;¥_1_RMB;”表示有两种货币，其中有一种货币:符号为HK$,相对于人民币的汇率是1.25，名字是叫HDK。在系统启动，载入配置文件时，我们根据该字符串生成多个对应的`Currency`对象，存在`EnvironmentContext`对象的一个list里。其中，`EnvironmentContext`类使用了单例模式，表示当前系统运行的环境上下文。

2. **饮料定制**

   2-1 推理和分析

饮料定制要求不同的系统可能有不同的定制饮料。我们小组成员讨论时，最初认为可以把这种定制饮料存在数据库中，不同店（系统）的数据库彼此独立。这样虽然可以满足需求，但是有些浪费，因为在咖啡销售系统的实际情况中，大多数饮料都是一样的。为了少量的定制饮料独立数据库，既浪费又不利于总部管理一些商品。毕竟受季节等因素影响，增删改某些商品是很常见的。因此，我们在讨论后认为，可以把定制饮料也放在配置文件里，在系统启动的时候读入即可。

2-2 总体设计

在配置文件中存放定制饮料的信息，在系统启动的时候，初始化环节，将定制饮料的信息读入，并存放在某个list里。在系统运行时，如果要访问饮品信息，就先从数据库中检索，若数据库中没有，就从存放定制饮料的list中检索。即把数据库和配置文件当作饮料信息的两个不同的源。

2-3 详细设计

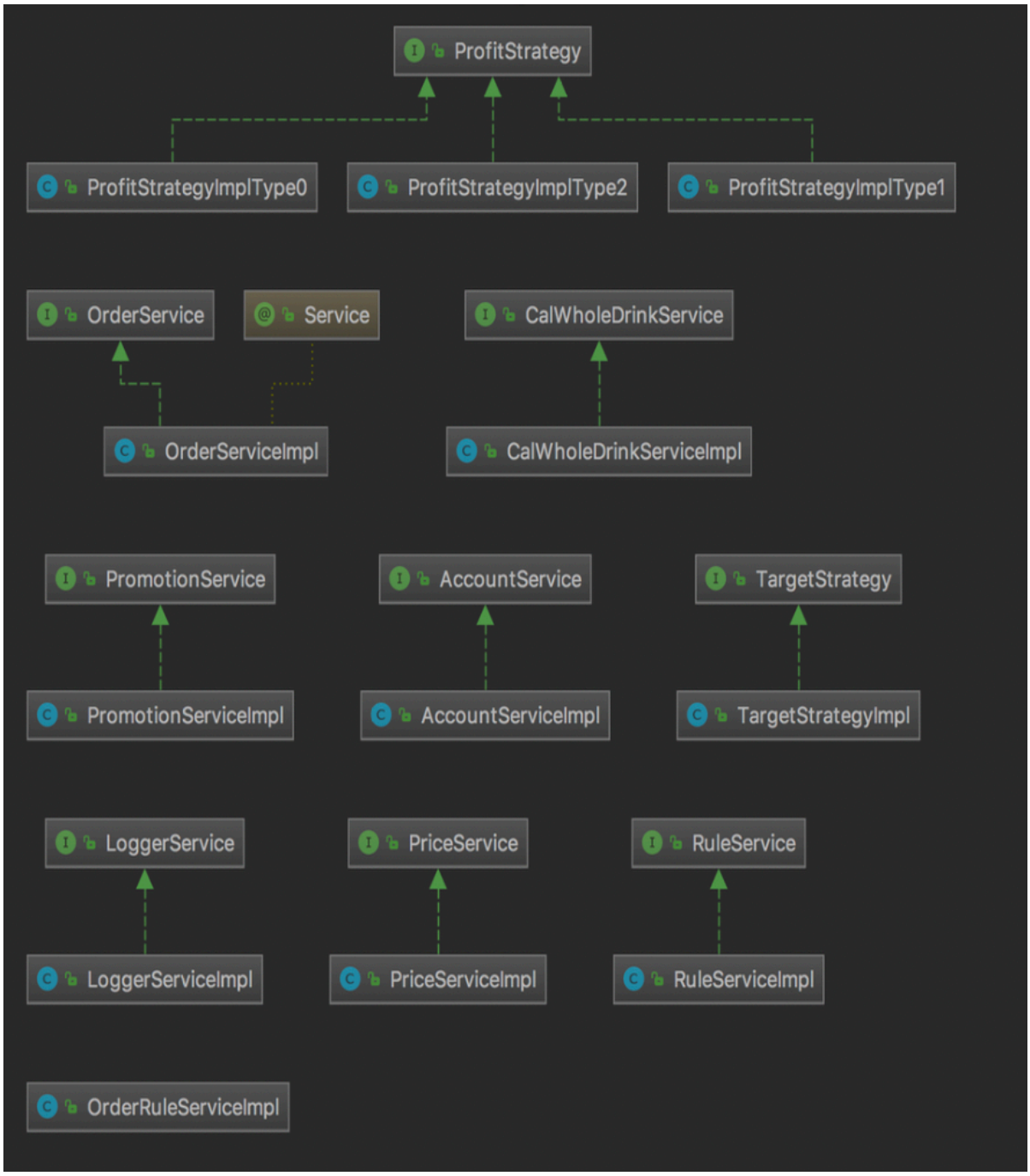
与货币的设计类似，我们统一用SPECIAL这个键表示定制饮料，键值设计成一个字符串。该字符串需要能表示多种定制饮料，每种饮料必需的信息。我们对于这个字符串的设计是：以分号分割不同的饮料，每一种饮料的不同信息以逗号分隔，顺序按照Drinks.csv中的顺序。比如，

`“cocoa,cocoa,20,1,3,6,9;wine,wine,18,1,2,4,5;”`表示有两种饮料，其中一种饮料：名字是cocoa，描述是cocoa，基础价格是20，默认size是1，小杯杯型价格是3，中杯杯型价格是6，大杯杯型价格是9。在系统启动，载入配置文件时，我们根据该字符串生成多个对应的Drinks对象，存在`EnvironmentContext`对象的一个list里。然后修改`DrinkRepositoryImpl`里的`getDrink`()方法，使得当在数据库中检索不到某饮料信息时，再去`EnvironmentContext`对象的存放定制饮料的list中检索。我们只修改了接口的实现，任意调用该接口访问饮料信息的代码都不会受影响。

3. **销售策略**

   3-1 推理和分析

   本次lab要求销售策略使用可插拔的方式进行添加和删除。而我们小组在lab4使用的是将促销规则存在数据库中的设计，也就是说，对销售策略的添加和删除是对数据库的添加和删除实现的。在咨询了助教后，认为数据库也算是可插拔的一种体现。所以，我们准备沿用lab4的设计，并在该基础上做一些完善，在一些环节加上策略模式，从而使整体设计能够符合“对于任意销售策略的插拔不会影响源码”。



注意，与存储饮料的表格不同，我们认为销售策略的数据库存储是不同店相互独立的，这样更方便修改和添加新的销售策略。我们会将指向 存储促销规则的表格 的路径放在配置文件里，这样有更大的灵活性，允许不同的数据库放置方式。

总体的设计分两部分，一是加入策略模式和反射机制，二是在lab4的基础上对rule的抽象的完善。

3-2 详细设计

加入策略模式和反射机制：lab4中，我们将 `ruleService` 的 `discount()` 函数用来判断某条促销规则对该订单是否生效，若生效又能给订单带来什么样的优惠。在lab5中，我们对这一函数重新进行了设计，将该过程分为三步。

第一步，资格判断，即判断客户的身份资格，如是否是VIP。这个在lab中没有要求，暂时没有实现。

第二步，目标判断，即判断这条规则对该订单现在是否生效。因为促销规则判断是否生效可能有不同的策略（如抽奖和满减的判断就完全不一样，抽奖收到全局订单数的制约，如中奖免单的只占总体数量的5%），所以我们在此处加入了策略模式，在符合开闭原则的前提下用不同的策略来判断。因为本次卓班的lab5只要求促销规则基于订单信息和系统信息（时间和地点），因此本次lab暂时只实现了基于这些信息的判断策略。

第三步，生效结果判断，即如果规则生效，具体优惠的情况。生效结果的判断方式是和促销规则的类型相关的，比如满减的优惠是数字的简单减少，打折的优惠是数字*折扣百分率，满赠的优惠是数字-饮料的价格，所以我们在此处也加入了策略模式，以三种不同的不同的策略对应三种促销规则的类型。同时，我们在往第三步加入策略模式的同时，也加入了反射机制。因为rule的`profitType`代表促销规则的类型，所以我们直接用`“ProfitStrategyImplType” + profitType`这种字符串拼接方式生成相应处理的策略类的名字。也就是说，一种类型的rule和一个策略是一一对应的。这样设计，在出现新类型的rule时，我们只需要加一个对应的策略类，不需要改动源代码。

在lab4的基础上对rule的抽象的完善：我们在lab4中对rule的抽象不合理，甚至无法表示lab5新加的两条销售策略。所以，在lab5中，我们对rule重新进行了设计。具体如下：

```
private int groupId; //优惠分组的组别
private int scope;  //资格范围，为0代表无限制，所有客户全都可以享受，这一部分可以进一步扩展成资格类型和资格配置
private int profitType; //利益类型,0是满减，1是满赠，2是打折
private double profit; //优惠 打8折就是0.8 送1杯就是1
private boolean canAdd;//是否支持累加
private Date from;
private Date to;
private int isOnlyBasicsDrinks;
private List<RuleRepositoryImpl.Item> discountRange;
private List<RuleRepositoryImpl.Item> orderCondition;
private List<RuleRepositoryImpl.Item> freeDrinks; //如果送饮料，送的类型/数量 不送的优惠为null
private String descirption;
```

在lab4的基础上

1）加入Date类型的from和to表示促销规则生效的时间范围。

2）对于促销规则生效地点的判断被省略了，因为促销规则是从本地数据库中读出，不是共享的。所以能读出的规则就代表是本店的促销规则。

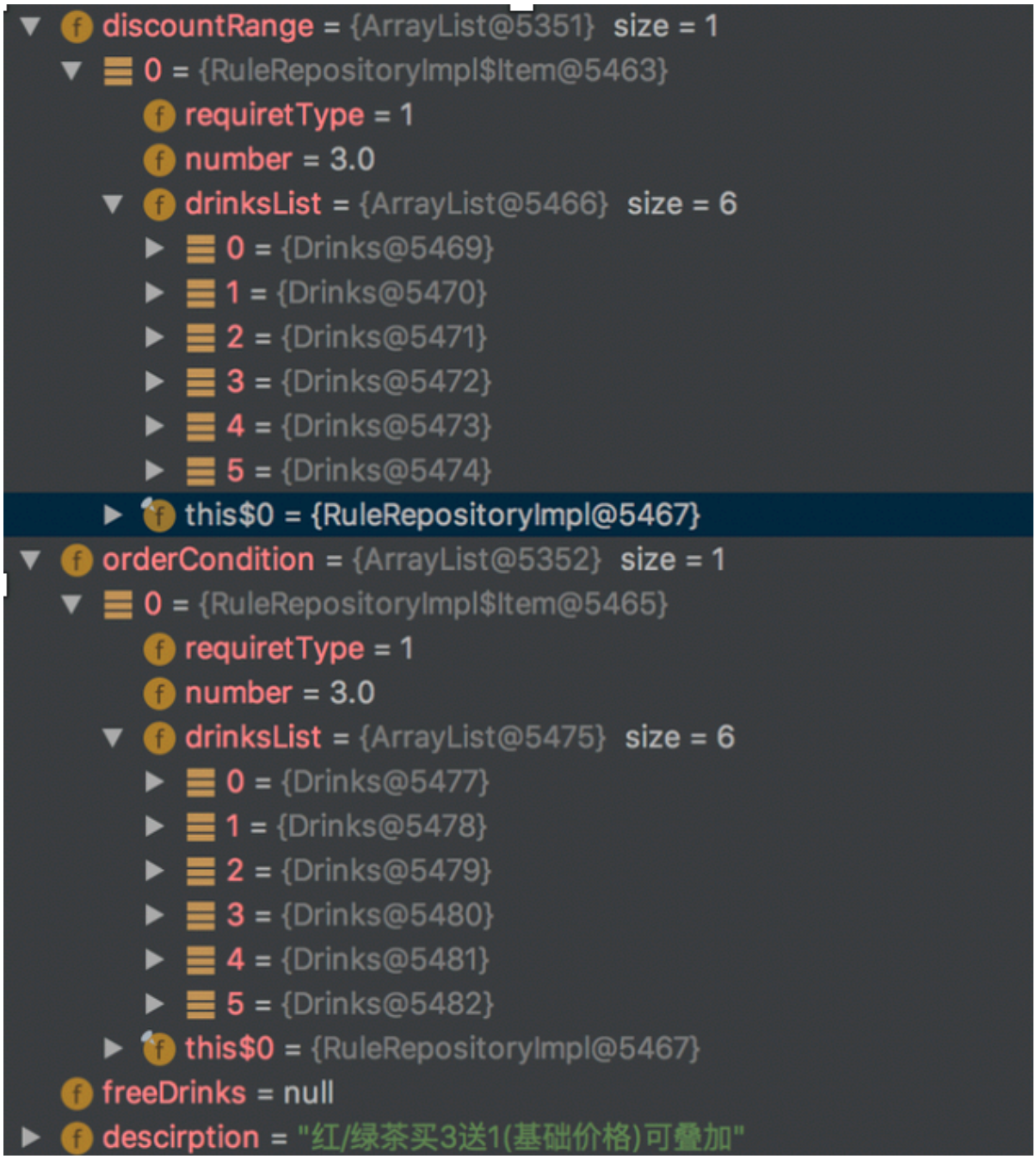
3）加入isOnlyBasicDrinks属性表示打折是对于整体价格、基础价格还是其它情况。

4）抽象一个Item对象，从数据库中存读，方便表示规则。Item对象如下：

```
protected int requiretType;//0价格 1杯数
protected double number;
protected List<Drinks> drinksList;
```

各属性的含义如上图注释，整体的语义就是“drinkList中的对象整体达到number，达到的判断来自requireType”。Drinklist里存的是对象而不是名字字符串，是因为扩展的需要，比如存取对象我们可以设置一些销售属性，那么我们就能满足“大杯红茶打八折”这种表述的需要。

5）在Item对象的基础上，抽出三个item的list来表示不同的信息：discountRange表示规则生效范围，orderCondition表示规则是否生效的判断条件，freeDrinks表示赠送饮料的信息。比如，对于“红/绿茶买3送1(基础价格)可叠加”这条促销规则，三个list中的情况如下：



其中，两个drinkList中存的都是6个对象：大中小杯的红茶，大中小杯的绿茶。

二、在详细设计的基础上，进一步阐述针对复杂工程问题的具体实现、测试（调试）以及结果分析

对TargetStrategy的开发测试与对OrderService的测试

- OrderService的测试主要分为两大类：第一类是对订单本身的测试，第二类是对促销方案策略的选择的测试。

```
声明：以下有关优惠信息列表中的内容，均用以下对应数字表示
1 = a 20% discount on every two Espresso(only big size)(basic-price)
2 = a free drink for every 3 cups of either red tea or green tea(basic-price)
3 = every two Cappuccino\uFF0Cthe second cup of half-price(basic-price)
4 = subtract 30 from every 100 for the whole order(whole-price)
5 = 50% off for all goods on 11/11(whole-price)
6 = when buying at least one cup of tea and one cup of coffee\uFF0C15% off for your order(whole-price)
```

1. 第一类包括对订单本身、订单中饮品列表、饮品标识符、饮品杯型、配料列表、配料标识符和配料份数以及不同饮品组合的测试，具体测试如下：
    - 订单为null：

输入的订单信息	期待的输出	期待的log
订单为null	运行时异常	The order has something wrong.

    - 对订单中饮品列表的长度的测试：

输入的订单信息	期待的输出	期待的log
饮品列表为null	运行时异常	The order has something wrong.
饮品列表中的长度为0	运行时异常	The order has something wrong.
饮品列表中只有一小杯espresso	原价=22，优惠=0，最终价格=22，优惠信息列表中无内容	无
饮品列表中有三小杯espresso	原价=66，优惠=0，最终价格=66，优惠信息列表中无内容	无

    - 对订单中饮品的name的测试：

输入的订单信息	期待的输出	期待的log
饮品name为null	运行时异常	Object not found, name: null
饮品name错误	运行时异常	Object not found, name: error
一杯饮品，name为espresso，无配料，小杯	原价=22，优惠=0，最终价格=22，优惠信息列表：[]	无
一杯饮品，name为cappuccino，无配料，小杯	原价=24，优惠=0，最终价格=24，优惠信息列表：[]	无
一杯饮品，name为redTea，无配料，小杯	原价=20，优惠=0，最终价格=20，优惠信息列表：[]	无
一杯饮品，name为greenTea，无配料，小杯	原价=18，优惠=0，最终价格=18，优惠信息列表：[]	无

- 对订单中饮品的杯型的测试：

输入的订单信息	期待的输出	期待的log
输入的订单信息	期待的输出	期待的log信息
饮品杯型小于1	运行时异常	无
饮品杯型大于3	运行时异常	无
一杯espresso，无配料，size为1	原价=22，优惠=0，最终价格=22，优惠信息列表：[]	无
一杯espresso，无配料，size为2	原价=24，优惠=0，最终价格=24，优惠信息列表：[]	无
一杯espresso，无配料，size为3	原价=26，优惠=0，最终价格=26，优惠信息列表：[]	无
一杯redTea，无配料，size为1	原价=20，优惠=0，最终价格=20，优惠信息列表：[]	无
一杯redTea，无配料，size为2	原价=22，优惠=0，最终价格=22，优惠信息列表：[]	无
一杯redTea，无配料，size为3	原价=23，优惠=0，最终价格=23，优惠信息列表：[]	无

- 对订单列表中不同饮品组合的测试：

输入的订单信息	期待的输出	期待的log
一小杯cappuccino+一小杯espresso	原价=46，优惠=0，最终价格=46，优惠信息列表：[]	无
一小杯redTea+一小杯greenTea	原价=38，优惠=0，最终价格=38，优惠信息列表：[]	无

- 对订单中配料列表的长度的测试：

输入的订单信息	期待的输出	期待的log
配料列表为null	运行时异常	Ingredient is invalid.
配料列表的内容为空，一小杯espresso	原价=22，优惠=0，最终价格=22，优惠信息列表：[]	无
配料列表中有一份份数为1的cream，一小杯espresso	原价=23，优惠=0，最终价格=23，优惠信息列表：[]	无
配料列表有三份份数为1的cream，一小杯espresso	原价=25，优惠=0，最终价格=25，优惠信息列表：[]	无

- 对配料name的测试：

输入的订单信息	期待的输出	期待的log
配料name为null	运行时异常	Ingredient is invalid.
配料name错误	运行时异常	Ingredient is invalid.
配料name为cream，一小杯加了一份cream的espresso	原价=23，优惠=0，最终价格=23，优惠信息列表：[]	无

- 对配料份数的测试：

输入的订单信息	期待的输出	期待的log
配料cream份数为-1，一小杯espresso	运行时异常	Ingredient is invalid.
配料cream份数为0，一小杯espresso	原价=22，优惠=0，最终价格=22，优惠信息列表：[]	无
配料cream份数为1，一小杯espresso	原价=23，优惠=0，最终价格=23，优惠信息列表：[]	无
配料cream份数为3，一小杯espresso	原价=25，优惠=0，最终价格=25，优惠信息列表：[]	无

- 对不同配料组合的测试：

输入的订单信息	期待的输出	期待的log
配料列表为一份milk+一份cream，一小杯espresso	原价=24.2，优惠=0，最终价格=24.2，优惠信息列表：[]	无

2. 第二类是对优惠策略选择的测试：
    - 对组合优惠中espresso的优惠方式的测试：

输入的订单信息	期待的输出
只有一大杯espresso	原价=26，优惠=0，最终价格=26，优惠信息列表：[]
一大杯espresso+一小杯espresso	原价=48，优惠=0，最终价格=48，优惠信息列表：[]
两大杯espresso，其中一杯加入一份cream	原价=53，优惠=8，最终价格=45，优惠信息列表：[1]
三大杯espresso	原价=79，优惠=8，最终价格=71，优惠信息列表：[1]

    - 对组合优惠中cappuccino的优惠方式的测试：

输入的订单信息	期待的输出
只有一小杯cappuccino	原价=24，优惠=0，最终价格=24，优惠信息列表中无内容
一大杯cappuccino+一小杯cappuccino	原价=52，优惠=11，最终价格=41，优惠信息列表：[3]
两小杯cappuccino，其中一杯加入一份cream	原价=49，优惠=11，最终价格=38，优惠信息列表：[3]
三杯cappuccino，分别是小杯，中杯，大杯	原价=78，优惠=11，最终价格=67，优惠信息列表：[3]
四小杯cappuccino	原价=96，优惠=22，最终价格=74，优惠信息列表：[3]

    - 对组合优惠中红茶的优惠方式的测试：

输入的订单信息	期待的输出
三小杯红茶	原价=60，优惠=0，最终价格=60，优惠信息列表：[]
四小杯红茶	原价=80，优惠=18，最终价格=62，优惠信息列表：[2]
九小杯红茶（此处测试九杯是因为5、6、7杯都会选择满100减30的优惠方式）	原价=180，优惠=36，最终价格=144，优惠信息列表：[2]
四杯红茶，三杯是小杯，一杯为大杯	原价=83，优惠=18，最终价格=65，优惠信息列表：[2]
四小杯，其中一杯加入一份cream	原价=81，优惠=18，最终价格=63，优惠信息列表：[2]

- 对组合优惠中绿茶的优惠方式的测试：

输入的订单信息	期待的输出
四小杯greenTea	原价=72，优惠=16，最终价格=56，优惠信息列表：[2]

- 对组合优惠中两种茶都有时的优惠方式的测试：

输入的订单信息	期待的输出
八小杯茶，其中一杯为红茶	原价=146，优惠=34，最终价格=112，优惠信息列表：[2]
八小杯茶，其中两杯为红茶	原价=148，优惠=36，最终价格=112，优惠信息列表：[2]
九小杯茶，其中一杯为红茶	原价=164，优惠=34，最终价格=130，优惠信息列表：[2]

- 对组合优惠中不同饮品优惠累加的测试:

输入的订单信息	期待的输出
两大杯espresso+六小杯cappuccino	原价=196，优惠=41，最终价格=155，优惠信息列表：[1，3]
八小杯greenTea+两小杯cappuccino	原价=192，优惠=43，最终价格=149，优惠信息列表：[2，3]
八小杯greenTea+两大杯espresso	原价=196，优惠=40，最终价格=156，优惠信息列表：[1，2]
两大杯espresso+4小杯greenTea+2小杯cappuccino	原价=172，优惠=35，最终价格=137，优惠信息列表：[1，2，3]

- 对满100减30的优惠方式的测试：

输入的订单信息	期待的输出
一小杯espresso	原价=22，优惠=0，最终价格=22，优惠信息列表：[]
一小杯espresso+一中杯espresso+三小杯greenTea	原价=100，优惠=30，最终价格=70，优惠信息列表：[4]
五小杯espresso	原价=110，优惠=30，最终价格=80，优惠信息列表：[4]
十小杯espresso	原价=220，优惠=60，最终价格=160，优惠信息列表：[4]

- 对满100减30和组合优惠两种优惠方式都有时最优优惠策略选择的测试：

输入的订单信息	期待的输出
五小杯redTea	原价=100，优惠=30，最终价格=70，优惠信息列表：[4]
七小杯greenTea+一小杯redTea	原价=146，优惠=34，最终价格=112，优惠信息列表：[2]
四小杯cappuccino+两大杯espresso	原价=148，优惠=30，最终价格=118，优惠信息列表：[1，3]

- 对时间的测试：

输入的订单信息	期待的输出
在优惠时间之内，一大杯espresso	原价=26，优惠=13，最终价格=13，优惠信息列表：[5]
不在优惠时间之内，一大杯espresso	原价=26，优惠=0，最终价格=26，优惠信息列表：[2]

- 对至少有一种茶和至少一种咖啡时的测试：

输入的订单信息	期待的输出
一小杯espresso+一小杯greenTea	原价=40，优惠=6，最终价格=34，优惠信息列表：[6]
两小杯espresso+两小杯greenTea	原价=80，优惠=12，最终价格=68，优惠信息列表：[6]

- TargetStrategy的测试

该类的用途主要就是判断订单是否满足某条rule的所有条件，如果满足，返回满足了几次，如果没有满足，返回-1

Rule的规定	订单	返回值
没有约束条件（具体场景可能是到店即送礼品）	三小杯红茶，原价为100	1
只对时间有约束（如双十一打折）	时间在优惠区间内	1
（同上）	时间不在优惠区间内	-1
订单总价达到100	原价为100	1
（同上）	原价为10	-1
要求某类饮品的总价达到100	三小杯红茶，原价为10	-1
至少三杯饮品（优惠类型为满减或者打折）	三小杯红茶	1
（同上）	两小杯红茶	-1
至少三杯饮品（优惠类型为满赠）	四小杯红茶	1
（同上）	三小杯红茶	-1
订单中至少3大杯红茶	三小杯绿茶	-1
（同上）	三大杯红茶	1
（同上）	三小杯红茶	-1
订单中至少3杯绿茶或红茶或卡布奇诺（都是小杯）	两小杯红茶+一小杯绿茶	1
（同上）	三杯espresso	-1
价格至少为100，可以叠加	原价为200	2
价格至少为100，不可以叠加	原价为200	1
至少有3杯饮品，可以叠加	九小杯红茶	3
至少有3杯饮品，不可以叠加	九小杯红茶	1
至少三杯饮品，其中至少一杯红茶	三小杯绿茶	-1
（同上）	两小杯红茶+两小杯绿茶	1
有时间区间限制，并且至少有3杯饮品	三小杯绿茶，时间合法	1
（同上）	三小杯绿茶，时间不合法	-1
（同上）	两小杯绿茶，时间合法	-1
优惠条件中的类型不合法	优惠条件中的类型不合法	-1

- TargetStrategy的具体实现：

该类的用途主要就是判断订单是否满足某条rule的所有条件，如果满足，返回满足了几次，如果没有满足，返回-1：

1. 判断是否有时间和订单中饮品种类或数量的要求，如果都没有，直接返回1
2. 调用内部私有的`isTimeValid()`方法判断时间是否在对应优惠规则的区间，如果规则的起始时间和结束时间都为null，直接返回最大整数，否则，如果在区间内，该方法返回1，不在返回-1
3. 调用内部的`isOrderConditionValid`方法判断订单中的饮品类型和数量是否满足条件，如果条件为null，直接返回最大整数，否则，根据其中每条要求的类型，调用相应方法，并返回最小满足要求的次数
4. 3中如果是对价格的前提条件，那么会调用`isPriceValid()`方法，如果订单的原价大于要求的最低价格，那么根据是否可以叠加返回相应的值，否则返回-1
5. 3中如果是对饮品的数量的要求，那么就会调用`isDrinksNumValid()`方法，如果规则中只要求了饮品数量，对种类无要求，根据是否可以叠加返回满足次数，否则遍历整个订单，查看其中满足条件的饮品总杯数，根据要求最小杯数和是否可以叠加返回满足次数，如果不满足条件，返回-1
6. 最终isValid返回的是2和3中得到的最小值

对ProfitStrategy的开发和测试

1. ProfitStrategyType0即满减这种优惠的处理测试：对应rule：满100减30

   测试函数为：`public RuleResult profitProcess(RuleContext ruleContext, Rule rule,int max)`

   RuleContext中含有order、rule以及订单不含任何优惠的总价purePrice

   Rule中含有优惠的规则

   max表示该优惠最多可以执行几次，不可叠加为1，不能执行该优惠为-1

   ruleCanAdd：true

   测试场景：用户总订单价格满100减30元可以叠加。

输入	期待输出
订单中含有3杯不同杯型的红茶 设置总价purePrice是100元 Rule为ruleCanAdd，可以叠加 max为1	运行正常， 其结果与相等 new RuleResult(ruleCanAdd, 30, "100-30 canAdd")
订单中含有3杯不同杯型的红茶 设置总价purePrice是250元 Rule为ruleCanAdd，可以叠加 max为2	运行正常， 其结果与相等 new RuleResult(ruleCanAdd, 60, "100-30 canAdd")
订单中含有3杯不同杯型的红茶 设置总价purePrice是250元 Rule为ruleNotAdd，不可以叠加 max为2	运行正常， 其结果与相等 new RuleResult(ruleCanAdd, 30, "100-30 canAdd")

2. ProfitStrategyType1即满赠这种优惠的处理测试：对应rule：红/绿茶买3送1（基础价格高的优先）可叠加

测试函数为：`public RuleResult profitProcess(RuleContext ruleContext, Rule rule,int max)`

RuleContext中含有order、rule以及订单不含任何优惠的总价purePrice

Rule中含有优惠的规则

max表示该优惠最多可以执行几次，不可叠加为1，不能执行该优惠为-1

rule:

ruleOneKind:表示绿茶支持买三送一（基础价格贵的优先）

ruleMoreKind:表示绿/红茶支持买三送一（基础价格贵的优先）

测试场景：用户总订单价格满100减30元可以叠加。

输入	期待输出
订单中含有3杯大杯绿茶 Rule为ruleOneKind max为1	运行正常， 其结果与相等 new RuleResult(ruleOneKind, 16, "onekind")
订单中含有3杯大杯绿茶 Rule为ruleMoreKind max为1	运行正常， 其结果与相等 new RuleResult(ruleMoreKind, 16, "morekind")
订单中含有2杯大杯绿茶和1杯红茶 Rule为ruleMoreAdd max为1	运行正常， 其结果与相等 new RuleResult(ruleMoreKind, 18, "morekind")
订单种含有4杯大绿茶和2杯红茶 Rule为ruleMoreAdd max为2	运行正常， 其结果与相等 new RuleResult(ruleMoreKind, 36, "morekind")

3. ProfitStrategyType2即打折

   测试函数为：`public RuleResult profitProcess(RuleContext ruleContext, Rule rule,int max)`

   RuleContext中含有order、rule以及订单不含任何优惠的总价purePrice

   Rule中含有优惠的规则

   max表示该优惠最多可以执行几次，不可叠加为1，不能执行该优惠为-1

   rule:

   `ruleAllKind`:对所有商品打折：包括购买一个茶或咖啡类时8.5折以及双十一5折

   `rulePriceCanAdd`:对某种商品价格打折:可叠加 考虑可打折的商品不是所有都存在

   `ruleKindCanAdd`:对某种特定（限定size）商品按杯数有优惠基础价格：大杯速溶咖啡八折，叠加

   `ruleKindNotAdd`:对某种特定（限定size）商品按杯数有优惠基础价格：大杯速溶咖啡八折，不可叠加，有第一次八折

   `ruleKindCanAdd2`:对某种商品按杯数有优惠基础价格：第二件半价可叠加

   `ruleKindCanAdd3`:对多种商品按杯数有优惠基础价格：有很多商品都可以半价优惠

   `ruleKindNotAdd2`:对某种商品按杯数有优惠：不可叠加，max会是1，只有第一个第二件半价会便宜

`ruleKindNotAdd3`:对多种商品按杯数有优惠基础价格：但是只有第一次符合的可以半价优惠

- 测试场景：

  `ruleAllKind`：总价打85折

  `rulePriceCanAdd`：对某类商品打折：这里举例的是茶这一类商品基础价格打8折。

输入	期待输出
订单中含有3杯中杯绿茶 Rule为ruleAllKind 总价85折 max为1	运行正常， 其结果与相等 new RuleResult(ruleAllKind, 15, "allKind")
订单中小杯浓缩咖啡，没有茶 Rule为rulePriceCanAdd max为1	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd, 0, "priceCanAdd")
订单中小杯绿茶 Rule为rulePriceCanAdd max为1	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd,3.2, "priceCanAdd")
订单种含有3杯红茶和4杯绿茶，基础价格打折 Rule为rulePriceCanAdd max为1	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd, 23.6, "priceCanAdd")
订单种含有3杯红茶和4杯绿茶以及中小杯特浓咖啡，基础价格打折 Rule为rulePriceCanAdd max为1	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd, 23.6, "priceCanAdd")

- 测试场景：

  `ruleKindCanAdd`：对某个特定商品优惠：大杯浓缩咖啡两杯八折

输入	期待输出
订单中含有小杯浓缩咖啡和小杯绿茶 Rule为ruleKindCanAdd max为1	运行正常， 其结果与相等 new RuleResult(ruleKindCanAdd, 0, "kindCanAdd")
订单中3杯大杯浓缩咖啡 Rule为ruleKindCanAdd max为1	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd, 8, "priceCanAdd")
订单中4杯大杯浓缩咖啡 Rule为ruleKindCanAdd max为2	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd, 16, "priceCanAdd")

- 测试场景：

  `ruleKindNotAdd`：对某个特定商品优惠：大杯浓缩咖啡两杯八折 不可叠加

输入	期待输出
订单中含有小杯浓缩咖啡和小杯绿茶 Rule为ruleKindNotAdd max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd, 0, "kindNotAdd")
订单中3杯大杯浓缩咖啡 Rule为ruleKindNotAdd max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd 8, "kindNotAdd")
订单中4杯大杯浓缩咖啡 Rule为ruleKindNotAdd max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd, 8 "kindNotAdd")

- 测试场景：

  `ruleKindCanAdd2`：对某类商品优惠：卡布奇诺第二杯半价

输入	期待输出
订单中含有3杯不同size的卡布奇诺 Rule为ruleKindCanAdd2 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindCanAdd2, 11, "kindCanAdd2")
订单中2杯小杯卡布奇诺1杯中杯卡布奇诺1杯大杯卡布奇诺 Rule为ruleKindCanAdd2 max为2	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd2, 22, "priceCanAdd2")
订单中2杯小杯卡布奇诺1杯中杯卡布奇诺1杯大杯卡布奇诺和一杯小杯浓缩咖啡 Rule为ruleKindCanAdd2 max为2	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd, 22, "priceCanAdd2")

- 测试场景：

  `ruleKindNotAdd2`：对某个特定商品优惠：大杯浓缩咖啡两杯八折 不可叠加

输入	期待输出
订单中含有3杯不同size的卡布奇诺 Rule为ruleKindNotAdd2 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd2, 11, "kindNotAdd2")
订单中2杯小杯卡布奇诺1杯中杯卡布奇诺1杯大杯卡布奇诺 Rule为ruleKindNotAdd2 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd2 11, "kindNotAdd2")
订单中2杯小杯卡布奇诺1杯中杯卡布奇诺1杯大杯卡布奇诺和一杯小杯浓缩咖啡 Rule为ruleKindNotAdd2 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd2, 11 "kindNotAdd2")

- 测试场景：

`ruleKindCanAdd3`：对多类商品优惠：大杯红茶和大杯绿茶都第二杯半价

输入	期待输出
订单中含有大杯红茶小杯卡布奇诺和浓缩咖啡 Rule为ruleKindCanAdd3 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindCanAdd3, 9, "kindCanAdd3")
订单中小杯卡布奇诺和浓缩咖啡七杯红茶六杯绿茶，其中3杯大杯红茶两杯大杯绿茶 Rule为ruleKindCanAdd3 max为1	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd3, 17, "priceCanAdd3")
订单中小杯卡布奇诺和浓缩咖啡七杯红茶六杯绿茶，其中3杯大杯红茶两杯大杯绿茶 Rule为ruleKindCanAdd3 max为2	运行正常， 其结果与相等 new RuleResult(rulePriceCanAdd3, 34, "priceCanAdd3")

- 测试场景：

  `ruleKindNotAdd3`：对多类商品优惠：大杯红茶和大杯绿茶都第二杯半价 不可叠加

输入	期待输出
订单中含有大杯红茶小杯卡布奇诺和浓缩咖啡 Rule为ruleKindNotAdd3 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd3, 9, "kindNotAdd3")
订单中小杯卡布奇诺和浓缩咖啡七杯红茶六杯绿茶，其中3杯大杯红茶两杯大杯绿茶 Rule为ruleKindNotAdd3 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd3 17, "kindNotAdd3")
订单中小杯卡布奇诺和浓缩咖啡七杯红茶六杯绿茶，其中3杯大杯红茶两杯大杯绿茶 Rule为ruleKindNotAdd3 max为1	运行正常， 其结果与相等 new RuleResult(ruleKindNotAdd3, 34，"kindNotAdd3")

4. 优惠策略具体实现：
   - ProfitStrategyImplType0实现：

     判断优惠策略类型：

     - 满100减30：

       满减的情况下，rule必定满足`rule.getDiscountRange().size() == 1 && rule.getDiscountRange().get(0).getDrinksList() == null`，默认是对订单总价进行满减，也就是默认`rule.getIsOnlyBasicsDrinks() == 0`，此时直接用`max * rule.getProfit()`，即得到优惠金额，当满减不可以叠加的时候就设置`max = 1`。
   - ProfitStrategyImplType1实现：

     判断优惠策略类型：

     - 红/绿茶买三赠一：

`rule.getDiscountRange() == null || rule.getDiscountRange().size() == 0`无条件赠送的情况暂不考虑，`rule.getFreeDrinks() != null`表示特殊的赠送规则，`rule.getFreeDrinks() == null`表示默认的赠送策略，买三送一，遍历orderItem将订单中商品种类和对应数量整理在`Map<String, Integer> drinkNum`中，遍历`discountRange`整理得到`Map<String, Integer> require`优惠商品范围的商品与数量映射，将其按照基础价格由高到低排序，`max = max * rule.getProfit()`得到最多赠送的杯数，按数量选取前面几杯（也就是最贵的饮品）赠送，得到正确的优惠

- ProfitStrategyImplType2实现：

  判断优惠策略类型：

  - 双十一半价以及购买至少一个茶或咖啡，订单总价打85折：

    `rule.getDiscountRange() == null`表示对订单中的全体对象进行打折优惠，`rule.getIsOnlyBasicsDrinks() == 0`表示对整体饮品的价格打折。此时只需要用订单总体价格乘以优惠率就可以得到优惠价格，否则表示对饮品基础价格打折。

  - 两杯大杯浓缩咖啡8折和卡布奇诺第二杯半价

    `rule.getDiscountRange() != null`表示对`discountrange`里的商品进行价格优惠，用下面的代码将订单中商品种类和对应数量整理在`Map<String, Integer> drinkNum`中，再对`discountrange`中的`processObject`对象的`drinklist`进行遍历，用`Map<String, Integer> require`记录所有优惠商品的范围，准备工作做好后开始打折。

    ```
     List<RuleRepositoryImpl.Item> condition =
    rule.getDiscountRange();
      //先记录订单中出现的所有饮品的数量
      Map<String, Integer> drinkNum = new HashMap<>();
      for (OrderItem orderItem : order.getOrderItems()) {
            String key = orderItem.getName() + "#" +
    orderItem.getSize();
            if (!drinkNum.containsKey(key)) {
                  drinkNum.put(key, 1);
            } else {
                  drinkNum.put(key, drinkNum.get(key) + 1);
            }
      }
    ```

    打折的话freeDrinks是一定为null的，开始打折，对discountRange即优惠商品范围进行遍历，打折分为两种：

    1. 一种是对优惠范围内所有商品进行打折，此时`processObject.getNumber() == 0`也就是对商品价格进行打折，默认对基础价格打折，判断此种优惠商品是否在订单内也就是drinkNum中是否含有该商品，有的话打折记录折扣。
    2. 另一种是对优惠范围内部分商品进行打折，包括两杯大杯浓缩咖啡八折，三杯大杯浓缩咖啡只能有2杯八折以及卡布奇诺第二杯半价。用`remain = max*processObject.getNumber()`得到这类商品最多能有几杯饮品优惠，遍历processObject.的drinklist（drinklist默认按照基础价格由高到低排序），取

remain和reqiure中饮品数量之和最小的那个杯数进行优惠打折，得到折扣。

对不同币种选择的开发和测试

`PriceServiceImpl`类

测试函数：`void changeCurrentCurrency(String name)`

- 测试场景：售货员将系统默认币种切换成了非当前地区支持的币种

输入	期待输出
name：THB （当前系统支持HDK，RMB）	抛出运行时异常， new RuntimeException(CURRENCY_NOT_SUPPORT)

测试函数：`double charge(double rmb, Currency currency)`

- 测试场景：售货员将系统默认币种切换成了港币

输入	期待输出
订单：一杯小杯cappuccino	运行正常， 其结果与相等 new PaymentInfo(30，0，30，空)
订单：一杯加1份奶油，1份牛奶的小杯espresso	运行正常， 其结果与相等 new PaymentInfo(30.25，0，30.25，空)
订单：3杯大Espresso 各加1份奶油	运行正常， 其结果与相等 new PaymentInfo(98.75，10，88.75，大杯浓咖啡两杯8折)
订单：大杯cappuccino，小杯cappuccino加1份奶油	运行正常， 其结果与相等 new PaymentInfo(66.25，13.75，52.5，两杯卡布奇诺第二杯半价)
订单：4杯小绿茶各加1份奶油	运行正常， 其结果与相等 new PaymentInfo(95，20，75，红绿茶买3送1)
订单：4杯小红茶各加1份奶油	运行正常， 其结果与相等 new PaymentInfo(105，30，75，买100减30)
订单：1杯小杯绿茶加1份奶油，1杯小杯浓咖啡加1份奶油	运行正常， 其结果与相等 new PaymentInfo(51.25，7.69，43.56，订单有1茶1咖啡全单85折)

三、在迭代开发过程中，对需求变更的应对措施，以及实现需求变更之后的经验总结

1. **加入定制化开发**

本次lab开发加入饮料定制、语言切换、货币切换这些定制化开发因素的需求，这在之前的lab中是无法做到的。之前的lab中，饮料、货币等都是写死在代码里的，没有提供切换的接口。在本次lab中，我们引入JAVA的`ResourceBundle`技术，将定制化开发因素写在配置文件中，在系统启动的时候载入。

2. **加入新的促销规则**

   本次lab新增了两条促销策略。然后我们小组分析发现，这两条策略在lab4中都无法很好的用lab4设计的rule对象表示。这就说明，我们lab4的抽象不太合理，并不能很好的表示所有促销规则。所以，在本次lab中，我们小组重新设计了rule对象，以满足现有的需求。

   同时，考虑到未来可能新增的优惠策略会有更多的限制条件，我们在系统初始化所有销售策略时，按照`salesrules.csv`文件反射出处理该csv文件的处理类来根据csv建立rule对象，也即是现有lab5的`data/salesrules.csv`文件存放的是可以满足lab5语言体系的（既存在订单条件，时间条件，地点条件的减/增/打折），通过该csv文件建立的rule满足“一条销售策略”最初始需要具备的条件：订单满足信息，时间信息，并能表达自己是减/增/打折还是其他更多的优惠形式。在此之后如果需要对rule对象引入更多的边界条件来判断优惠策略，可以生成rule的子类，并将新rule抽象之后放入新的`salesrules2.csv`中，再实现该csv文件可以反射的`RepositoryImpl`类，在此类中写生成rule子类的方式，就可以新加上这条新的优惠策略了，且不需要更改代码。

3. **可插拔组件的开发**

   本次lab要求使用可插拔组件增加和删除销售策略。我们lab4用的是数据库来存取促销规则，本身就是可插拔的。但是，不同类型的促销规则要有不同的解析方式。所以，我们在解析过程加入了策略模式，并且利用反射机制来为每种类型的rule映射处理策略。

4. **经验总结**

   4-1

   在lab4到lab5的需求变更中，我们认为值得庆幸的一点是，我们可以在lab4的基础上优化扩展，而不是完全重构。我认为这是因为我们lab4考虑到了可扩展性，在设计的时候使得销售策略本身是易于修改、易于增加、易于删除的。

   4-2

   我们在lab4中试图从很简单的几条销售规则抽象出一个很完善的rule对象，然而这个设计完全不能覆盖lab5。所以我们在lab5的扩展中，让rule可能存在子类，允许生成不同类型的rule对象。这样的话，不会因为rule对象抽象的不好而导致在之后的lab需要再重构rule对象。

   4-3

   我们理解到其实任何抽象都不一定可以满足未来的条件，都需要对未来可能的需求增加尽可能多的可扩展性。如果整个系统只能像lab4中通过将rule抽象成一种样子来增减销售策略，在当时环境中也许非常简单便捷，但是从长远角度考虑可能反而会降低可扩展性。所以在优化系统设计架构的同时需要注意这一点。

四、阐述解决复杂工程问题所需工具的学习和使用情况，阐述相关文献、书籍的查阅、分析、总结和收获情况

1. **wiki的使用**

在复杂工程项目开发中，使用文档进行交流比微信聊天或者口头阐述更加明确、清晰。所以我们小组学习了wiki的写法，在项目开发过程中同时更进wiki的注解，使得组员间的交流更加流畅，代码组合之后的bug大大减少。

在查阅相关内容时，我们发现，大型项目中使用wiki是非常常见和必要的行为，wiki是一份包含项目进度，分工，对项目架构、各个接口的设计描述，注意事项，个人对项目的理解等项目相关信息的，可以多人合作撰写的文档。我组在制作wiki时也参考了这些内容，特别对自己项目的service实现，DAO对象，DTO对象设计，数据库表设计，销售规则的抽象形式进行了详细的描述，这种描述使得在开发中避免了一些因沟通、理解错误可能会导致的矛盾。使我组在完成了一个相对较复杂的“优惠策略实现”设计之后，能比较迅速和高正确率的完成实际的开发。同时，也为每个组员提供了理解他人代码的途径，也为开发者本身记录了自己的实现逻辑，为未来回看代码带去便利。

我们的wiki：

https://devcloud.huaweicloud.com/wiki/project/f5d92ce432954d64b7e658edc9961367/wiki/view/doc/335292

2. **原型开发**

   经过资料的查阅和整理，我们认为原型是一个工程项目最根本的架构，是产品最初子集的完整功能实现，是被设计好后应该可以很好适应增减新功能而不大改架构的“项目样机”。它提供了系统主要的功能和接口，主要价值是强化沟通，降低风险，节省后期变更成本，提高项目成功率。

   在查阅中，我们发现，有一些人所认为的“原型”，不是具体的初期代码，而可能是一张可视化的图片，绘制了项目大概的架构。在我们的学习中，我们觉得“原型“既可以是可视化的图纸，提供接口、类、数据结构的设计（比如我们wiki文档在项目开始时最先编辑的那部分），而更重要的是它应该是本次项目的一个”框架“，它展现了项目的需求，体现了项目为了完成需求采用的一些设计模式，也含有一种扩展性，在原型上通过一系列小规模开发循环后可以完成整个项目的开发。所以本次项目开发，使用的也是基于lab4的原型进行的开发，从最重要的优惠策略来讲，我们的计算架构的设计在lab4上已经非常完善，但是在lab5中新增的促销策略有的却无法被抽象成和lab4上同样的形式，在此基础上我们重写了对规则的抽象模式，试图使其更加契合各种情况，同时也对未来可能有的同样无法被抽象成lab5中形式的促销规则增加另外处理的接口，使得lab5的原型在lab4的基础上更加健壮和更加框架化。

3. **可插拔组件**

   经过查找学习后，我们大致认为，可插拔，即插上拔下都不影响系统正常运行，也就是说，在增加已有同类型功能新需求的时候（比如店里多支持一种货币/多一种促销策略），可以很快速的适应新需求而不影响原型，同样在删除已有同类型功能需求时可以不影响原型系统运行的删除。

   原型为项目提供的是类似框架的功能，可插拔的原型设计是希望开发相应需求的时候，可以使用原型，而不用改变原型就可以实现需求。

   我们认为，最符合可插拔的设计的就是“通过增加配置系统自动实现新功能需求”，比如增加配置文件，系统就满足了需要新加的功能。为了在多语言/货币/策略的基础上达到这样的目标，我们设计了一个自认为比较完善的架构，在上文已经阐述了，同时这个架构也具有很高的拓展性，通过修改配置文件里特定的参数，系统在运行过程中会自动反射指定的类来处理，这个类可以是后写的处理类，但是加入这个类的过程并不需要改变原型的代码，整个过程非常自动和可插拔。

4. **java反射类**

   我们学习并在项目中使用了反射机制来提高原型的可扩展性和一些组件的插拔性。

对于反射的实现，总结为：1.反射不仅仅是用类名来反射的，更详细的说，用的其实是路径来反射的，所以在包里的类反射是需要带上包里的路径才能在反射时找到该类。2.反射可以直接抽取指定类的指定的构造函数，并通过构造函数生成该类，好处是为类初始化了参数（带参构造函数） 3.反射内部类时，idea中名称格式应该为“外表类名$内部类名”的形式，因为这是内部类被编译后的名称，同时内部类的构造参数会比能看见的参数多出一个参数，用来记录外部类。

反射是一种很方便的类的调用，它的使用大大提高了我们代码的可扩展性。

五、分析、总结和归纳项目执行过程中存在的主要问题，基于具体案例说明应对项目风险与挑战的能力培养情况

- **主要问题：**
  1. **架构设计与代码实现过程中的沟通**

     因为我们的整个项目架构比较复杂，有非常多的接口、类，以及rule对象的抽象模式相对较为复杂，不同形式表达含义会非常不同，这就在打好架构后，代码实现的同学会对已打好的框架的功能和rule对象的含义的理解产生疑惑，我组解决此问题的方式是写文档说明以及在项目中需要进行实现的位置增加TODO来告知。

  2. **配置文件中特殊符号乱码**

     我们在使用ResourceBundle在配置文件中读入的¥，中文逗号的时候会多出乱码比如Â等，是因为ResourceBundle使用时按unicode读入。解决方法是将文件中的中文和特殊符号设置成\uXXXX\uXXXX的形式，或者在读入前使用`getbytes（“ISO-8859-1”,“UTF-8”）`来规定按照UTF-8读入。

- **具体案例说明应对项目风险与挑战的能力培养情况：**
  1. **（项目风险）未来如何在不修改原型的基础上增减支持货币种类，改变语言环境，增减销售策略**

     为了能够契合以上的要求，我们设想的是通过抽象出上述各内容与同类对象中的共同点，存入配置文件/csv文件中，使得之后有上述需求时，可以简单的通过在配置文件/csv文件里增删内容来达到效果，也是我们的一种可插拔的实现方式。

     为了实现这一点，我们非常努力的学习了如何抽象共同点，规划了很多service实现与DTO传输对象，在写对rule的处理时尽可能想到了对后续更多情况的处理。

  2. **(项目风险）未来加入的销售策略如果添加了新属性而不能被抽象成lab5中已有的模式**

     我们期望的是未来的尽可能多的按照已有语言体系书写的销售策略可以不增加/删除一行代码地加入系统的计算，这种已有的语言体系指：在优惠条件包含店铺地点信息，时间信息，订单满足条件信息时，采用减/送/打折的优惠。凡是在这个语言体系下的销售策略，都可以通过在现有的salesrules.csv里增减文字内容而不增减任何代码来达到策略的增减。

     同时，也必然要考虑到如果优惠方式不止减/送/打折的情况，我们的rule对象里存了一个值表明自己是减/送/打折对象，如果新的策略超出了这3种优惠方式，则可以通过修改此值，并在Strategy包的impl包下增加一个名称为`“ProfitStrategyImplType+该值”`的类来对rule进行处理，返回discount和message信息即可。

最后，考虑到未来可能有新的策略是当前rule抽象无法表示完全的，所以我们通过反射csv表对应的`ruleRepositoryImpl`来生成rule对象。在这样的设计下，如果有新的策略需要加入新的量（比如从lab4到lab5对rule加入了时间的限制）时，可以将新rule抽象放入新的一个csv文件中，并对应这个csv文件写一个新的rule读取对象，这个过程不需要变动仍和已有代码，只需要新加入csv文件和对应的rule对象实现类即可。

综上，我们的销售策略组件是非常可插拔的。

小组成员与分工

账号	姓名	学号	分工
achillessanger	石睿欣	17302010065	架构设计，货币切换
HXX5656	胡宵宵	17302010077	架构设计
song	宋怡景	17302010079	测试，优惠策略实现
zyhlx	张逸涵	17302010076	测试，优惠策略实现