

第二次 “软件系统架构设计” 课程实践大作业

——2020090910022 王莹

针对一个在线点餐外卖系统，给出该系统的下列软件功能模块详细设计。

1. 在线点餐外卖系统的订单功能模块设计类图设计
2. 在线点餐外卖系统的订单功能模块顺序图设计
3. 在线点餐外卖系统的订单功能模块通信图设计
4. 在线点餐外卖系统的订单功能模块状态图设计
5. 在线点餐外卖系统的订单功能模块代码框架设计
6. 在线点餐外卖系统的订单功能设计模式优化设计与 Java 编程

作业要求：采用专业建模设计工具给出各个模型设计图，并给出设计思想、设计问题探讨、设计说明。

作业文件格式：作业 2_学号_姓名.doc\作业 2_学号_姓名. 模型文件

作业成绩评价标准：

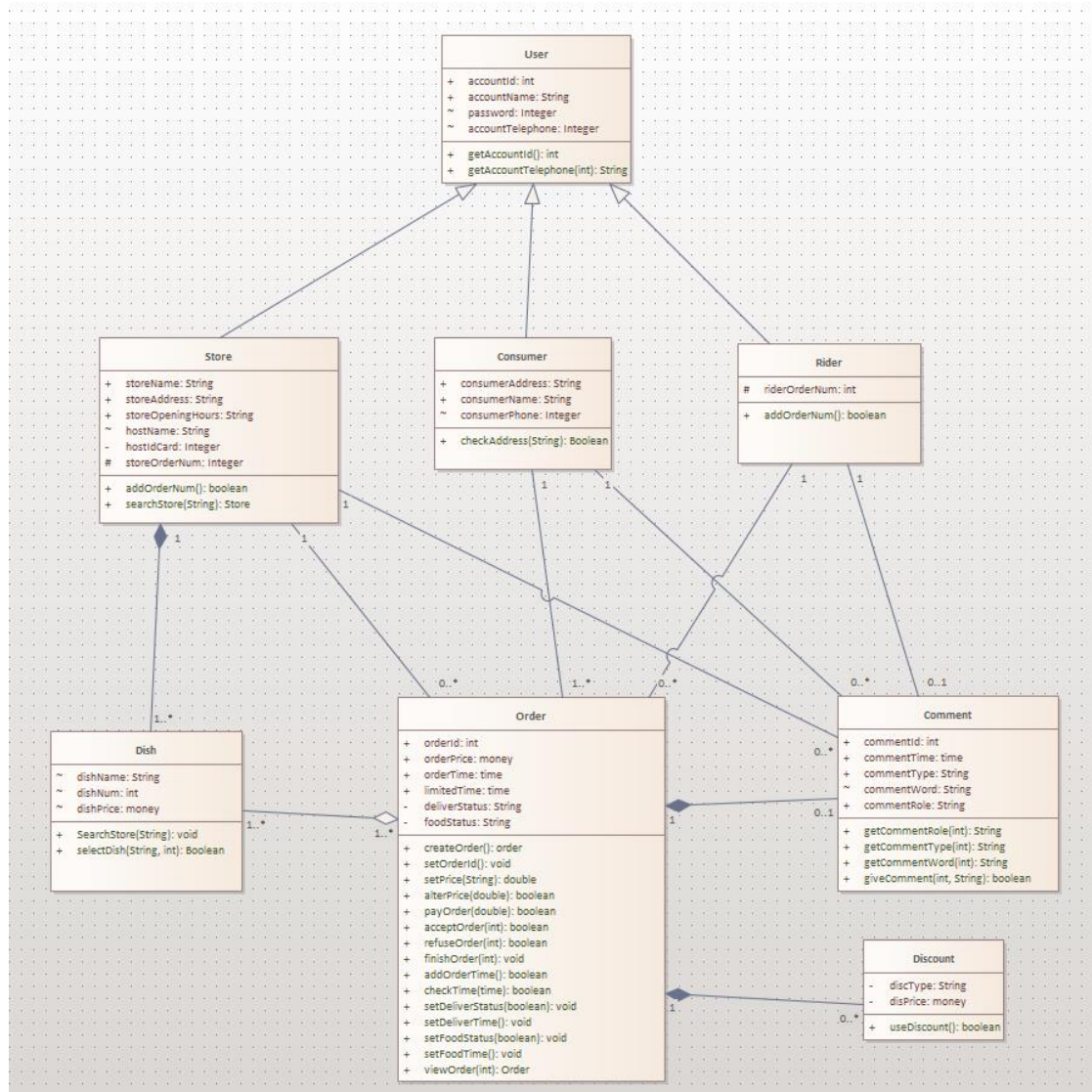
正确完成情况	优	优	优	良	良	良	良	良	中	中
作业过程情况	优	优	良	优	良	良	良	中	中	中
文档规范性	优	良	良	优	优	良	中	中	中	差
作业评分	100-98	97-95	94-92	91-89	88-85	84-82	81-79	78-76	75-73	72-70

目录

一、 设计类图设计.....	3
1. 类图设计图.....	3
二、 顺序图设计.....	4
1. 用户查看订单顺序图.....	4
2. 顾客搜索商家顺序图.....	5
3. 顾客下单顺序图.....	6
4. 顾客提交订单顺序图.....	7
5. 骑手处理订单顺序图.....	8
6. 商家处理订单顺序图.....	10
三、 通信图设计.....	11
1. 用户查看订单通信图.....	12
2. 顾客搜索商家通信图.....	12
3. 顾客下单通信图.....	13
4. 顾客提交订单通信图.....	13
5. 骑手处理订单通信图.....	14
6. 商家处理订单通信图.....	15
四、 状态图设计.....	15
1. 订单状态图.....	16
2. 商家状态图.....	16
五、 代码框架设计.....	17
1. User 类.....	17
2. Consumer 类.....	18
3. Store 类.....	19
4. Rider 类.....	20
5. Comment 类.....	21
6. Order 类.....	22
7. Dish 类.....	25
8. Discount 类.....	26
六、 设计模式优化设计与 Java 编程.....	27
1. 单例模式优化.....	27
2. 桥接模式优化.....	29
3. Java 编程.....	29
1) Order 类.....	29
2) Store 类.....	33
3) viewCommentDemo 类.....	34
4) ConsumerDemo 类.....	36
5) RiderDemo 类.....	38
6) StoreDemo 类.....	40

一、设计类图设计

1. 类图设计图



图表 1 设计类图

2. 设计思想

类图设计图是在分析类图基础之上对系统分析类图的细化与完善设计，从而在设计层面描述系统的静态结构。设计类图需要详细定义各个类的属性、方法与类之间关系的设计约束。建立类图需要确定类的属性与类之间的关系；类的方法根据系统动态模型确定与完善。

3. 设计问题探讨

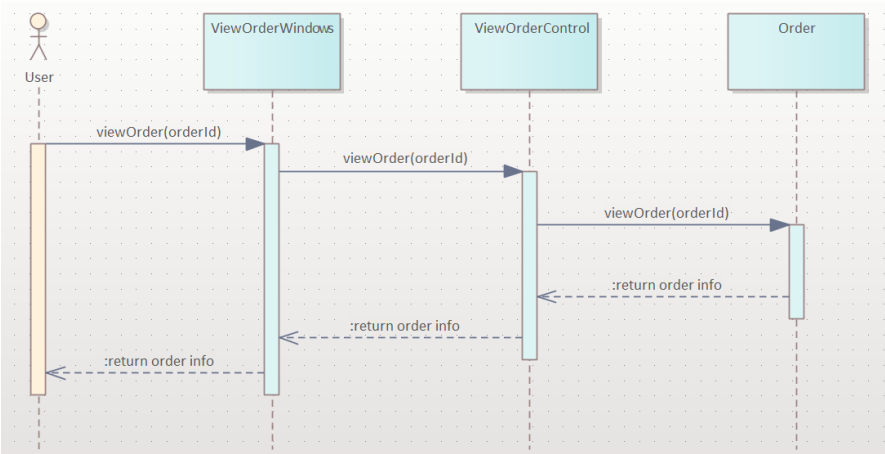
在此类图中，我确定了 User（用户）、Consumer（顾客）、Rider（骑手）、Store（商家）、Order（订单）、Dish（菜品）、Discount（优惠券）、Comment（评价）八个主要业务实体类。其中 Consumer、Rider、Store 为参与者，系统需要存储订单信息，需要 Book 类；需要存储菜品，需要 Dish 类；需要存储订单中使用的优惠，需要 Discount 类；需要存储订单的评价，需要 Comment 类。各个类的属性如图所示，方法根据顺序图得到。

二、顺序图设计

设计思想：顺序图按照从上至下顺序描述对象之间的消息交互，目的是展示功能逻辑的实现；顺序图由一组对象及他们之间的消息组成，强调消息的先后时间顺序。

1. 用户查看订单顺序图

1) 顺序图



图表 2- 1 用户查看顺序图

2) 设计问题探讨

在本顺序图中，首先要确定交互对象为：User、Order, 顾客需要与系统之间进行交互，所以需要查看界面 ViewOrderWindow、界面需要通过查看控制对象 ViewOrderControl 来控制业务逻辑, 然后再添加对象之间的交互消息。

3) 设计说明

User 是参与者，作为发起者放在顺序图的最左边，ViewOrderWindow 是边界类对象，ViewOrderControl 是控制类对象，Order 是实体类对象。

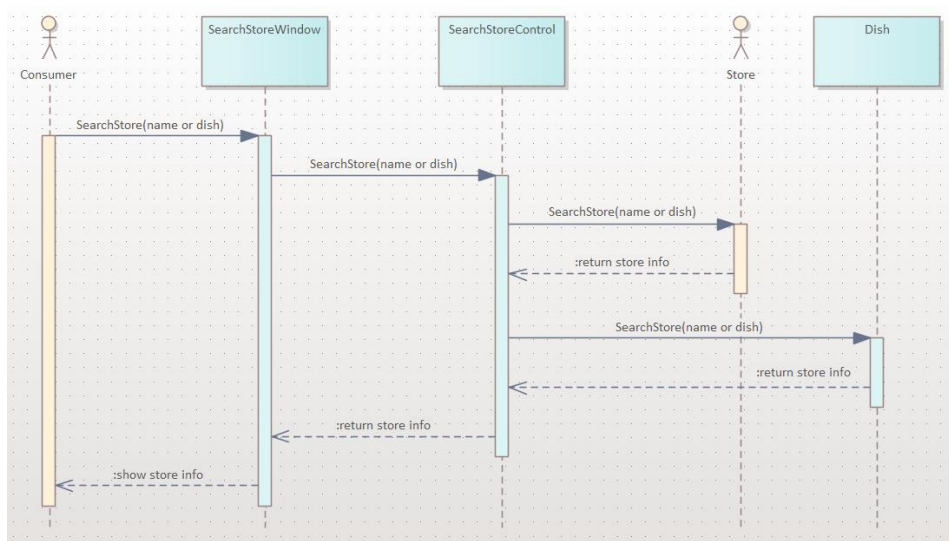
它们的交互过程如下：

顾客通过查看界面输入订单 ID 请求查看订单；查看界面根据订单 ID 向查看控制请求查看订单；查看控制订单 ID 请求订单信息；查看控制将订单信息返回给查看界面；查看界面将订单信息显示给用户。

根据交互，可以发现 Order 类需要提供 viewOrder(orderId)操作。

2. 顾客搜索商家顺序图

1) 顺序图



图表 2- 2 顾客搜索商家数据库

2) 设计问题探讨

在本顺序图中，首先要确定交互对象为：Consumer、Store、Dish，顾客需要与系统之间进行交互，所以需要搜索界面 SearchStoreWindow、界面需要通过搜索控制对象 SearchStoreControl 来控制业务逻辑，然后再添加对象之间的交互消息。

3) 设计说明

Consumer 是参与者，作为发起者放在顺序图的最左边，SearchStoreWindow 是边界类对象，SearchStoreControl 是控制类对象，Store、Dish 是实体类对象。

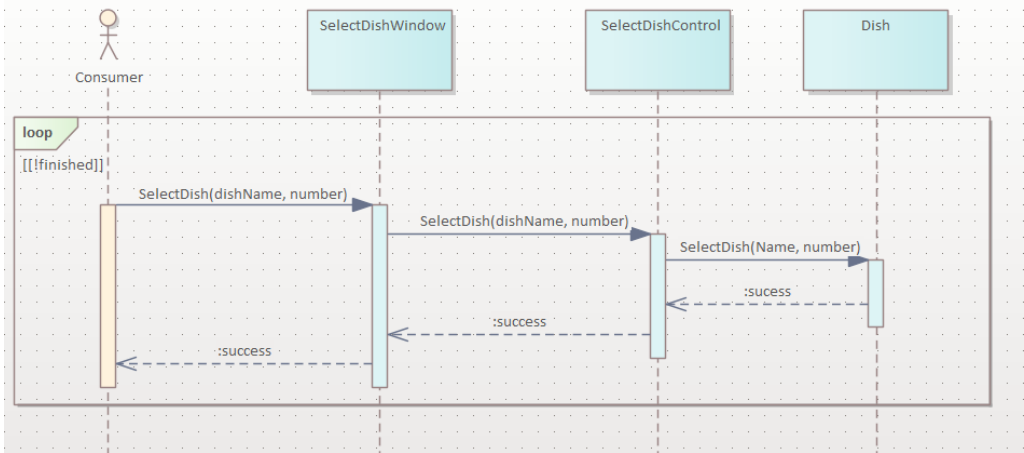
它们的交互过程如下：

顾客通过搜索界面输入店铺或菜品名称请求搜索；搜索界面根据输入信息向搜索控制请求商家信息与菜品信息；搜索控制根据输入信息请求商家信息；搜索控制根据输入信息请求菜品信息；搜索控制将商家与菜品信息返回给搜索界面；搜索界面将信息显示给顾客。

根据交互，可以发现 Store 类需要提供 searchStore(storeName) 操作；Dish 类需要提供 searchStore(storeName) 操作。

3. 顾客下单顺序图

1) 顺序图



图表 2- 3 顾客下单顺序图

2) 设计问题探讨

在本顺序图中，首先要确定交互对象为：Consumer、Dish，顾客需要与系统之间进行交互，所以需要菜品选择界面 SelectDishWindow、菜品选择界面需要通过菜品选择控制对象 SelectDishControl 来控制业务逻辑, 然后再添加对象之间的交互消息。

3) 设计说明

Consumer 是参与者，作为发起者放在顺序图的最左边，SelectDishWindow 是边界类对象，SelectDishControl 是控制类对象，Dish 是实体类对象。

它们的交互过程如下：

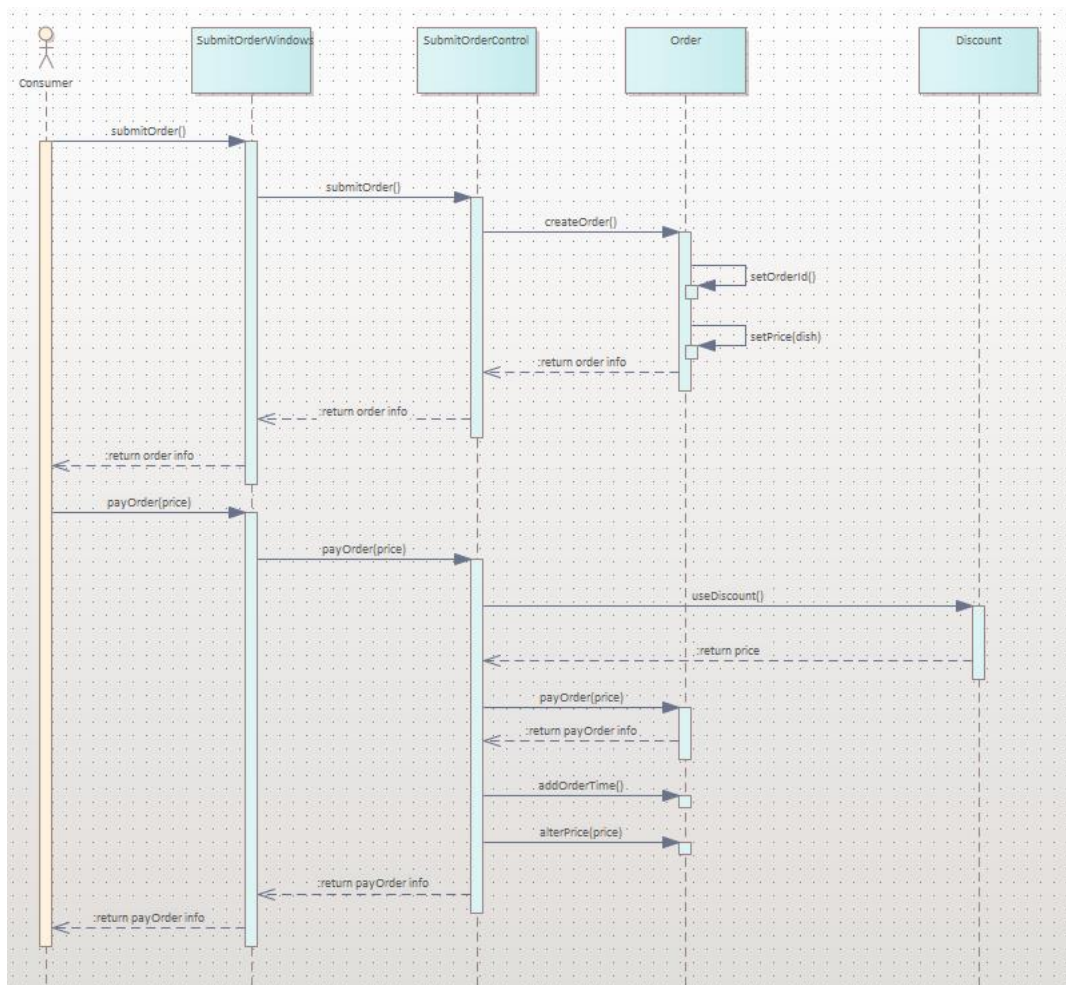
如果顾客不选择结束，则顾客可以完成如下循环：通过菜品选择界面输入菜品名称与数量请求选择菜品；菜品选择界面根据菜品名称与数

量向菜品选择控制请求选择菜品；菜品选择控制请求选择菜品；菜品选择控制将成功信息返回给菜品选择界面；菜品选择界面将成功信息显示给顾客。

根据交互，可以发现 Dish 类需要提供 SelectDish(Name, number) 操作。

4. 顾客提交订单顺序图

1) 顺序图



图表 2- 4 顾客提交订单顺序图

2) 设计问题探讨

在本顺序图中，首先要确定交互对象为:Consumer、Order、Discount，顾客需要与系统之间进行交互，所以需要提交界面 SubmitOrdeWindow、界面需要通过提交控制对象 SubmitOrderControl 来控制业务逻辑,然后

再添加对象之间的交互消息。

3) 设计说明

Consumer 是参与者，作为发起者放在顺序图的最左边，SubmitOrderWindow 是边界类对象，SubmitOrderControl 是控制类对象，Order、Discount 是实体类对象。

它们的交互过程如下：

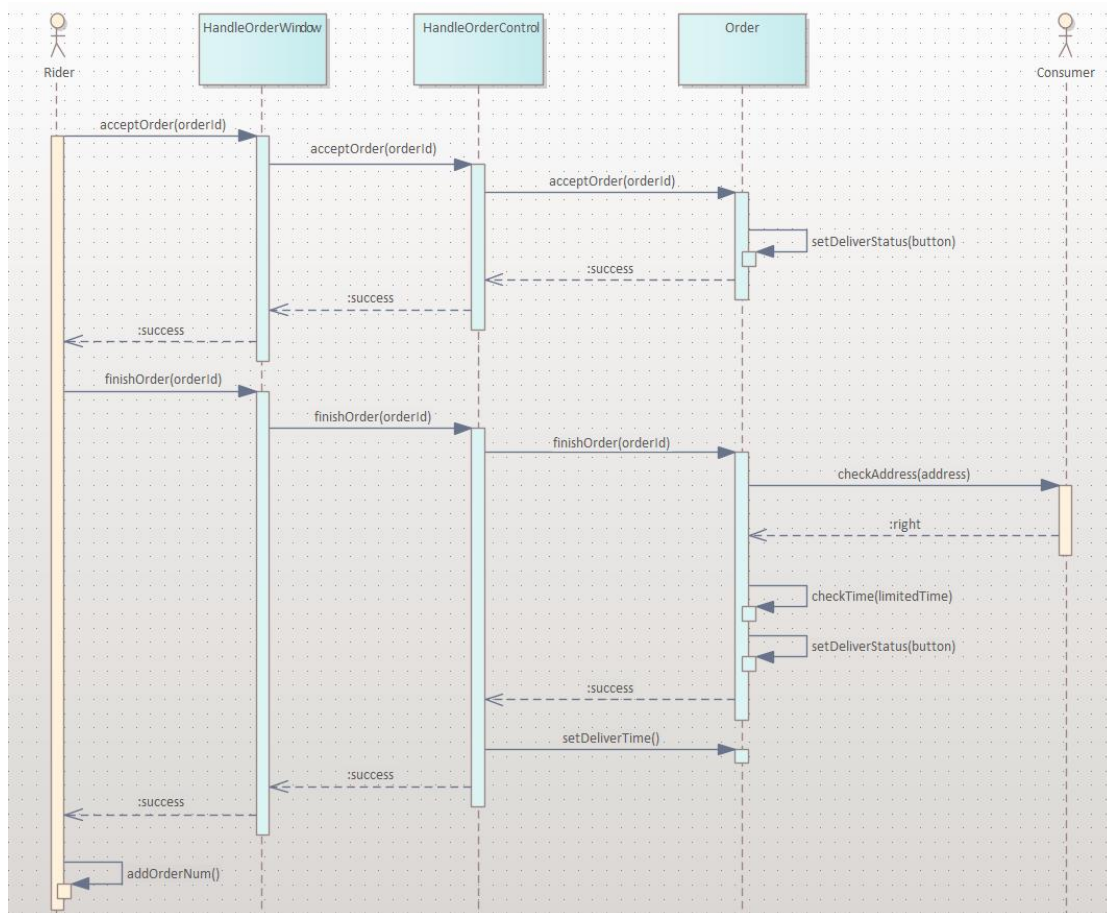
顾客通过提交界面请求提交订单；提交界面向提交控制请求提交订单；提交控制请求创建订单；订单设置订单 ID 与订单价格；提交控制将订单信息返回给提交界面；提交界面将订单信息显示给顾客。

顾客通过提交界面请求支付订单；提交界面向提交控制请求支付订单；提交控制请求使用优惠，成功则返回修改后的价格给提交控制；提交控制请求支付订单；提交控制请求增加订单提交时间；提交控制请求更改订单价格；提交控制将支付信息返回给提交界面；提交界面将支付信息显示给顾客。

根据交互，可以发现 Order 类需要提供 createOrder()、setOrderId()、setPrice(dish)、payOrder(price)、addOrderTime()、alterPrice(price)操作；Discount 类需要提供 useDiscount()操作。

5. 骑手处理订单顺序图

1) 顺序图



图表 2- 5 骑手处理订单顺序图

2) 设计问题探讨

在本顺序图中，首先要确定交互对象为：Rider、Order，顾客需要与系统之间进行交互，所以需要处理订单界面 HandleOrderWindow、界面需要通过处理订单控制对象 HandleOrderControl 来控制业务逻辑，然后再添加对象之间的交互消息。

3) 设计说明

Rider 是参与者，作为发起者放在顺序图的最左边，HandleOrderWindow 是边界类对象，HandleOrderControl 是控制类对象，Order 是实体类对象。

它们的交互过程如下：

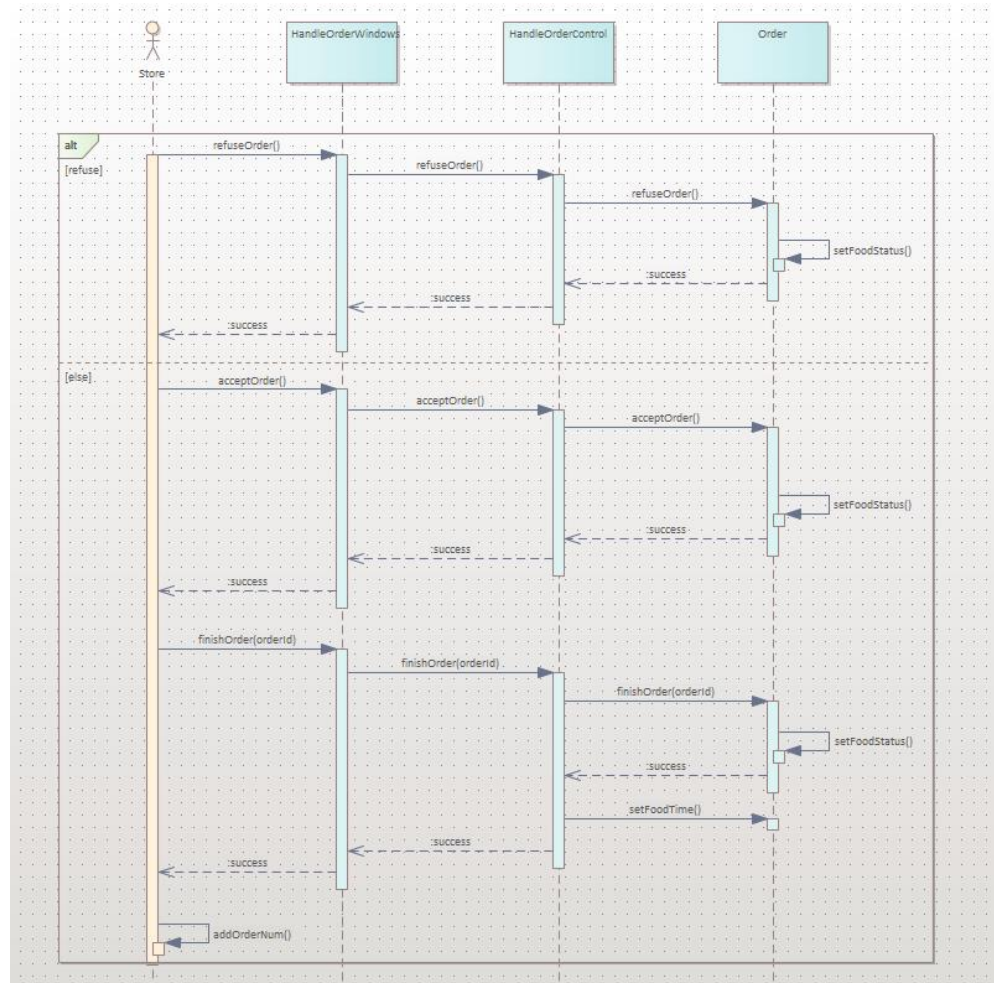
骑手通过处理订单界面输入订单 ID 请求接收订单；处理订单界面根据输入订单 ID 向处理订单控制请求接收订单；处理订单控制请求接收订单；订单设置订单配送状态；处理订单控制将成功信息返回给处理订单界面；处理订单界面将成功信息显示给骑手。

骑手将订单送达后，通过处理订单界面输入订单 ID 请求完成订单；处理订单界面根据订单 ID 向处理订单控制请求完成订单；处理订单控制请求完成订单；订单检查骑手是否到达顾客位置；订单检查配送时间是否超时；订单设置配送状态；处理订单控制请求设置订单配送完成时间；处理订单控制将成功信息返回给处理订单界面；处理订单界面将成功信息显示给骑手。骑手请求增加完成订单单数。

根据交互，可以发现 Order 类需要提供 acceptOrder(orderId)、setDeliverStatus()、finishOrder(orderId)、checkAddress(consumerAddress)、checkTime(limited) 操作；Rider 类需要提供 addOrderNum() 操作。

6. 商家处理订单顺序图

1) 顺序图



图表 2- 6 商家处理订单顺序图

2) 设计问题探讨

在本顺序图中，首先要确定交互对象为：Store、Order，顾客需要与系统之间进行交互，所以需要处理订单界面 HandleOrderWindow、界面需要通过处理订单控制对象 HandleOrderControl 来控制业务逻辑，然后再添加对象之间的交互消息。

3) 设计说明

Store 是参与者，作为发起者放在顺序图的最左边，HandleOrderWindow 是边界类对象，HandleOrderControl 是控制类对象，Order 是实体类对象。

它们的交互过程如下：

(1) 如果商家拒绝，则商家通过处理订单界面请求拒绝订单；处理订单界面向处理订单控制请求拒绝订单；处理订单控制请求拒绝订单；订单设置备餐状态；处理订单控制将成功信息返回给处理订单界面；处理订单界面将成功信息显示给商家。

(2) 否则商家通过处理订单界面请求接收订单；处理订单界面向处理订单控制请求接收订单；处理订单控制请求接收订单；订单设置备餐状态；处理订单控制将成功信息返回给处理订单界面；处理订单界面将成功信息显示给商家。

(3) 商家完成备餐后，通过处理订单界面请求完成订单；处理订单界面向处理订单控制请求完成订单；处理订单控制请求完成订单；订单设置备餐状态；处理订单控制请求设置备餐完成时间；处理订单控制将成功信息返回给处理订单界面；处理订单界面将成功信息显示给商家。商家请求增加完成订单单数。

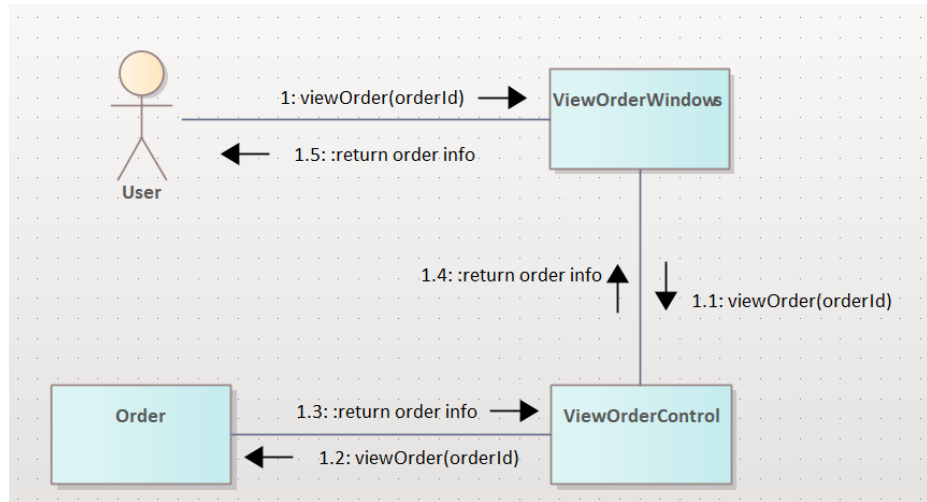
根据交互，可以发现 Order 类需要提供 acceptOrder(orderId)、refuseOrder(orderId)、setFoodStatus()、finishOrder(orderId) 操作；Store 类需要提供 addOrderNum() 操作。

三、通信图设计

设计思想：通信图也是交互图的一种，与顺序图不同的是，顺序图按照时间布图，通信图按照空间布图。通信图由参与交互的对象和它们之间的链组成，链上附着消息，消息的时间顺序使用序号表示。在线点餐外卖系统的通信图也有 6 个，如下所示。

1. 用户查看订单通信图

1) 通信图



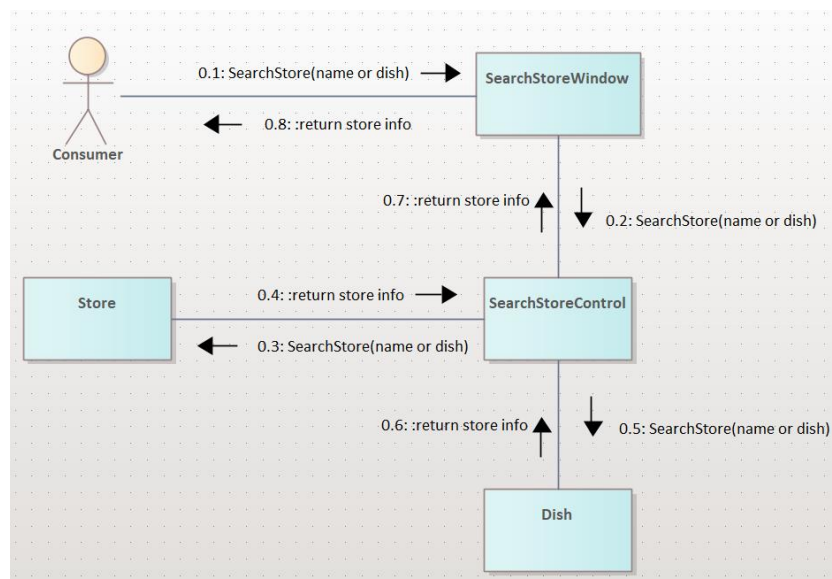
图表 3- 1 用户查看订单通信图

2) 设计说明

通信图的设计与顺序图相对应，确定交互对象 User、Order, 搜索界面 ViewOrderWindow、搜索控制对象 ViewOrderControl；序号、消息与顺序图的消息交互对应，将各个对象根据交互连接，并添加共 6 条消息。

2. 顾客搜索商家通信图

1) 通信图



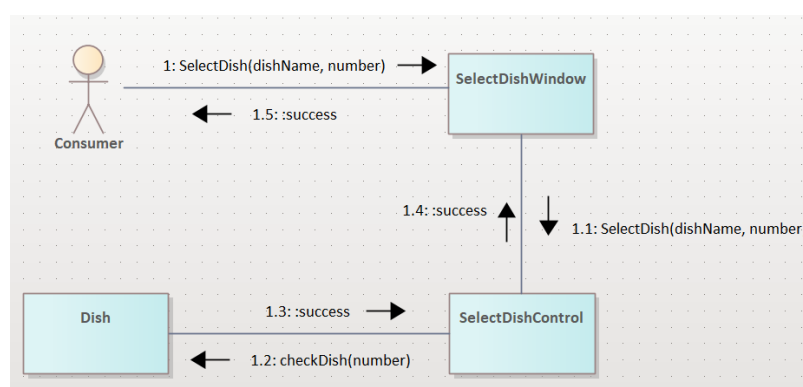
图表 3- 2 顾客搜索商家通信图

2) 设计说明

通信图的设计与顺序图相对应，确定交互对象 Consumer、Store、Dish，搜索界面 SearchStoreWindow、搜索控制对象 SearchStoreControl；序号、消息与顺序图的消息交互对应，将各个对象根据交互连接，并添加共 8 条消息。

3. 顾客下单通信图

1) 通信图



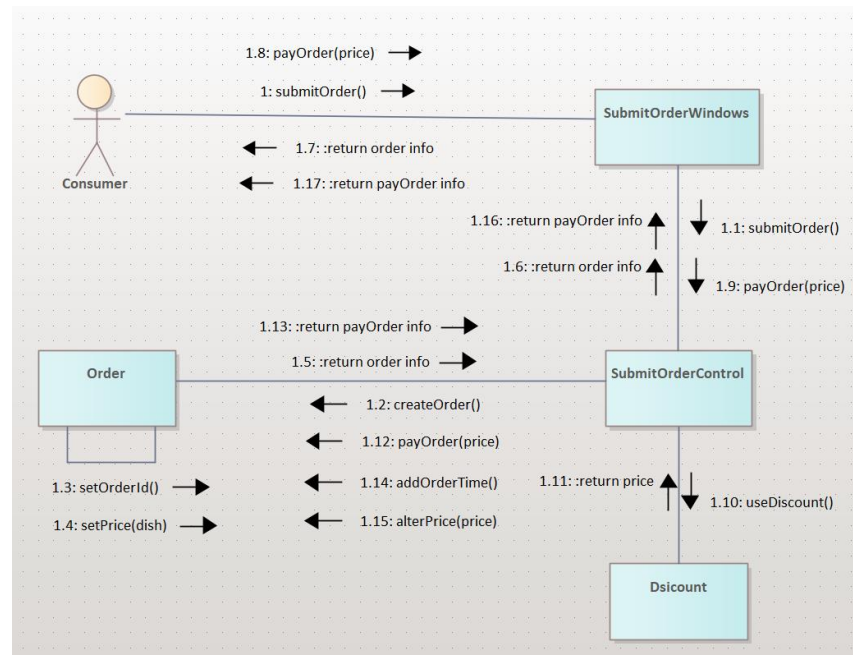
图表 3- 3 顾客下单通信图

2) 设计说明

通信图的设计与顺序图相对应，确定交互对象 Consumer、Store、Dish，搜索界面 SearchStoreWindow、搜索控制对象 SearchStoreControl；序号、消息与顺序图的消息交互对应，将各个对象根据交互连接，并添加共 6 条消息。

4. 顾客提交订单通信图

1) 通信图



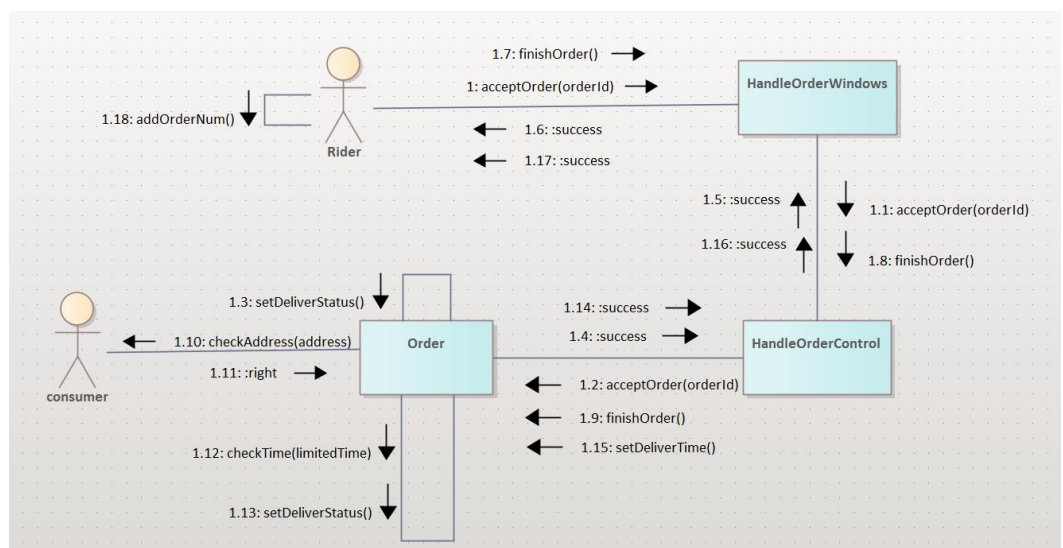
图表 3- 4 顾客提交订单通信图

2) 设计说明

通信图的设计与顺序图相对应，确定交互对象 Consumer、Discount、Order，搜索界面 SubmitOrderWindow、搜索控制对象 SubmitOrderControl；序号、消息与顺序图的消息交互对应，将各个对象根据交互连接，并添加共 18 条消息。

5. 骑手处理订单通信图

1) 通信图



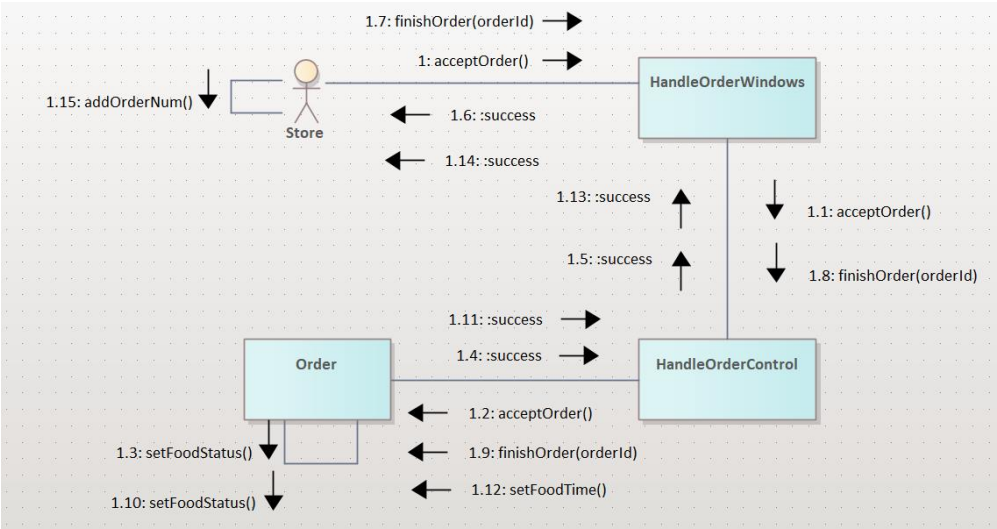
图表 3- 5 骑手处理订单通信图

2) 设计说明

通信图的设计与顺序图相对应，确定交互对象 Rider、Order, 搜索界面 HandleOrderWindow、搜索控制对象 HandleOrderontrol；序号、消息与顺序图的消息交互对应，将各个对象根据交互连接，并添加共 19 条消息。

6. 商家处理订单通信图

1) 通信图



图表 3- 6 商家处理订单通信图

2) 设计说明

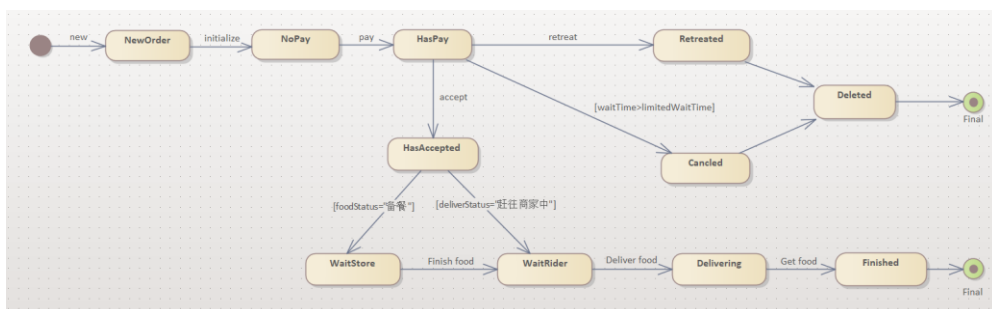
通信图的设计与顺序图相对应，确定交互对象 Store、Order，搜索界面 HandleOrderWindow、搜索控制对象 HandleOrderControl；序号、消息与顺序图的消息交互对应，将各个对象根据交互连接，并添加共 16 条消息。

四、状态图设计

设计思想：状态机图通常依附于类，对系统中重要的类进行建模，表明该类的对象的详细行为。在在线点餐外卖系统中，我们选择订单类创建状态机图。

1. 订单状态图

1) 状态图



图表 4- 1 订单状态机图

2) 设计问题探讨

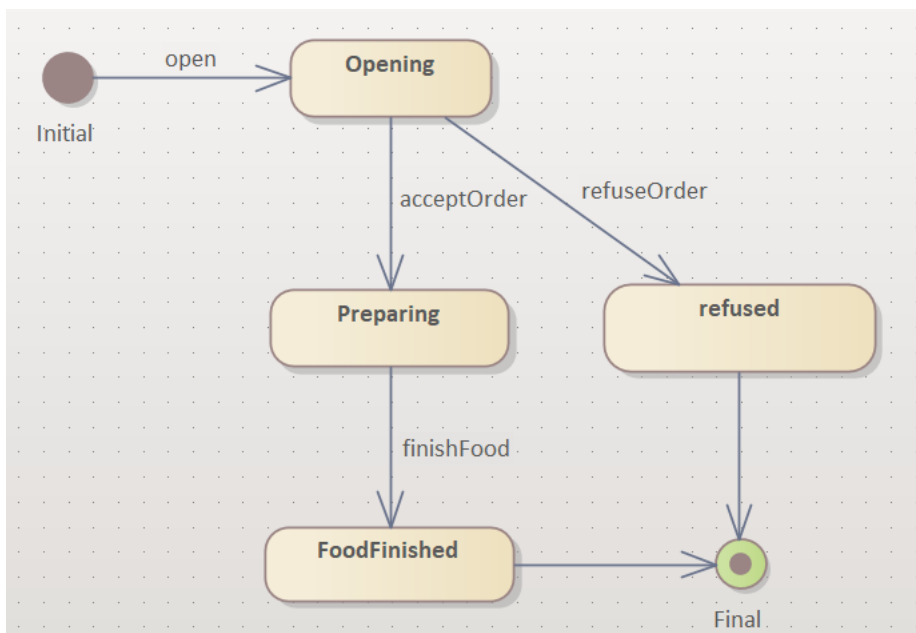
Order 类可以有如下状态：新订单(NewOrder)、未支付(NoPay)、已付款(HasPay)、撤回(Retreated)、被取消(Canceled)、被删除(Deleted)、已被接单(HasAccepted)、商家未出餐(WaitStore)、骑手未取餐(WaitRider)、配送中(Delivering)、已完成(Finished)。

3) 设计说明

顾客提交订单后，在线点餐外卖系统会新建订单，Order 处于新订单的状态；而后系统对其进行初始化，Order 处于未支付状态；顾客在支付订单后，Order 变为已支付状态；此时顾客可以选择撤回订单，Order 进入撤回状态，而后被系统自动删除；若在系统规定的等待时间内，商家或骑手未接单，Order 被取消，被系统自动删除；商家和骑手接单后，Order 进入已被接单状态；若菜品状态(FoodStaus)为“备餐”，则 Order 处于商家未出餐的状态；若商家已完成备餐或配送状态(DeliverStatus)为“赶往商家中”，则 Order 处于骑手未取餐的状态；骑手取餐后开始配送，Order 进入到配送状态；顾客收到外卖后，订单完成。

2. 商家状态图

1) 状态图



图表 4- 2 商家状态机图

2) 设计问题探讨

Store 类可以有如下状态：开业(Opening)、备餐(Peraring)、备餐完成(FoodFinished)、拒绝(refused)。

3) 设计说明

商家开始营业后，商家处于开业状态；当顾客提交订单后，商家若选择接收订单，商家则进入备餐状态，备餐完成后，进入备餐完成状态；若商家拒绝订单，则进入拒绝状态，订单结束。

五、代码框架设计

设计思想：使用 EA 软件，将详细类图导出生成 Java 代码。

1. User 类

```

package 设计类图;

/**
 * @author 王莹
 * @version 1.0
 * @created 26-11 月-2022 18:17:38
  
```

```

*/
public class User {

    public int accountId;
    public String accountName;
    Integer password;
    Integer accountTelephone;

    public User() {

    }

    public void finalize() throws Throwable {

    }

    public int getAccountId() {
        return 0;
    }

    /**
     *
     * @param accountId
     */
    public String getAccountTelephone(int accountId) {
        return "";
    }
} //end User

```

2. Consumer 类

```

package 设计类图;

/**
 * @author 王莹
 * @version 1.0
 * @created 26-11 月-2022 18:17:38

```

```

    */
public class Consumer extends User {

    public String consumerAddress;
    public String consumerName;
    Integer consumerPhone;
    public Order m_Order;
    public Comment m_Comment;

    public Consumer() {

    }

    public void finalize() throws Throwable {
        super.finalize();
    }
    /**
     *
     * @param address
     */
    public Boolean checkAddress(String address) {
        return false;
    }
} //end Consumer

```

3. Store 类

```

package 设计类图;

/**
 * @author 王莹
 * @version 1.0
 * @created 26-11 月-2022 18:17:38
 */
public class Store extends User {

    public String storeName;
    public String storeAddress;
    public String storeOpeningHours;
    String hostName;
    private Integer hostIdCard;
    protected Integer storeOrderNum;

```

```

    public Order m_Order;
    public Comment m_Comment;
    public Dish m_Dish;

    public Store() {

    }

    public void finalize() throws Throwable {
        super.finalize();
    }
    public boolean addOrderNum() {
        return false;
    }

    /**
     *
     * @param name or dish
     */
    public Store searchStore(String name or dish) {
        return null;
    }
} //end Store

```

4. Rider 类

```

package 设计类图;

/**
 * @author 王莹
 * @version 1.0
 * @created 26-11 月-2022 18:17:38
 */
public class Rider extends User {

    protected int riderOrderNum;
    public Order m_Order;
    public Comment m_Comment;

    public Rider() {

```

```

    }

    public void finalize() throws Throwable {
        super.finalize();
    }

    public boolean addOrderNum() {
        return false;
    }
} //end Rider

```

5. Comment 类

```

package 设计类图;

/**
 * @author 王莹
 * @version 1.0
 * @created 26-11 月-2022 14:07:39
 */
public class Comment {

    public int commentId;
    public time commentTime;
    public String commentType;
    String commentWord;
    public String commentRole;
    public commentRole m_commentRole;

    public Comment() {

    }

    public void finalize() throws Throwable {

    }
    /**
     *
     * @param orderId
     */
    public String getCommentType(int orderId) {
        return "";
    }
}

```

```

    }

    /**
     *
     * @param commendId
     */
    public String getCommentWord(int commendId) {
        return "";
    }

    /**
     *
     * @param commentRole
     */
    public boolean giveComment(String commentRole) {
        return false;
    }

    /**
     *
     * @param orderId
     */
    public String getCommentRole(int orderId) {
        return "";
    }

    /**
     *
     * @param commentId
     * @param s
     */
    public boolean giveComment(int commentId, String s) {
        return false;
    }
} //end Comment

```

6. Order 类

```

package 设计类图;

/**
 * @author 王莹

```



```

* @version 1.0
* @created 26-11 月-2022 18:17:38
*/
public class Order {

    public int orderId;
    public money orderPrice;
    public time orderTime;
    public time limitedTime;
    private String deliverStatus;
    private String foodStatus;
    public Discount m_Discount;
    public Dish m_Dish;
    public Comment m_Comment;

    public Order() {

    }

    public void finalize() throws Throwable {

    }
    public order createOrder() {
        return null;
    }

    public void setOrderId() {

    }

    /**
     *
     * @param dish
     */
    public double setPrice(String dish) {
        return 0;
    }

    /**
     *
     * @param price
     */
    public boolean alterPrice(double price) {
        return false;
    }

```

```

}

/**
 *
 * @param price
 */
public boolean payOrder(double price) {
    return false;
}

/**
 *
 * @param orderId
 */
public boolean acceptOrder(int orderId) {
    return false;
}

/**
 *
 * @param orderId
 */
public boolean refuseOrder(int orderId) {
    return false;
}

/**
 *
 * @param orderId
 */
public void finishOrder(int orderId) {

}

public boolean addOrderTime() {
    return false;
}

/**
 *
 * @param limitedTime
 */
public boolean checkTime(time limitedTime) {
    return false;
}

```

```

    }

    /**
     *
     * @param button
     */
    public void setDeliverStatus(boolean button) {

    }

    public void setDeliverTime() {

    }

    /**
     *
     * @param button
     */
    public void setFoodStatus(boolean button) {

    }

    public void setFoodTime() {

    }

    /**
     *
     * @param orderId
     */
    public Order viewOrder(int orderId) {
        return null;
    }
} //end Order

```

7. Dish 类

```

package 设计类图;

/**
 * @author 王莹
 * @version 1.0

```

```

* @created 26-11 月-2022 18:17:38
*/
public class Dish {

    String dishName;
    int dishNum;
    money dishPrice;

    public Dish() {

    }

    public void finalize() throws Throwable {

    }
    /**
     *
     * @param name or dish
     */
    public void SearchStore(String name or dish) {

    }

    /**
     *
     * @param Name
     * @param number
     */
    public Boolean selectDish(String Name, int number) {
        return false;
    }
} //end Dish

```

8. Discount 类

```

package 设计类图;

/**
 * @author 王莹
 * @version 1.0
 * @created 26-11 月-2022 18:17:38
 */
public class Discount {

```

```

private String discType;
private money disPrice;

public Discount() {

}

public void finalize() throws Throwable {

}

public boolean useDiscount() {
    return false;
}

} //end Discount

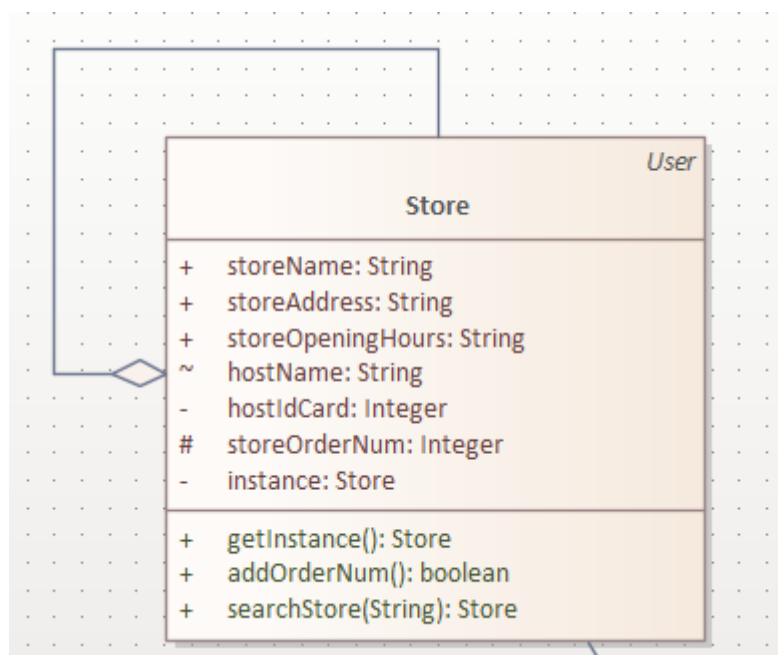
```

六、设计模式优化设计与 Java 编程

1. 单例模式优化

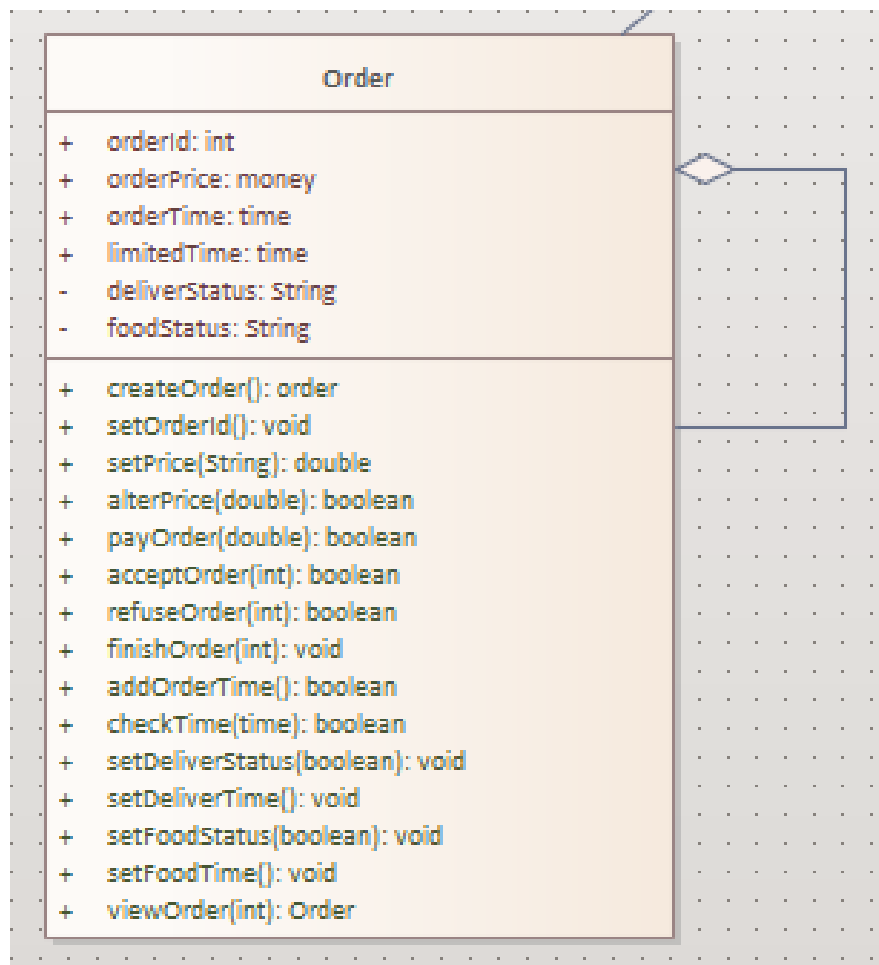
设计思想：Order 类与 Store 类需要频繁创建，在创建时开销较大，且需要频繁访问，可以通过单例模式解决多实例带来的系统开销问题。

1) Store 单例模式优化



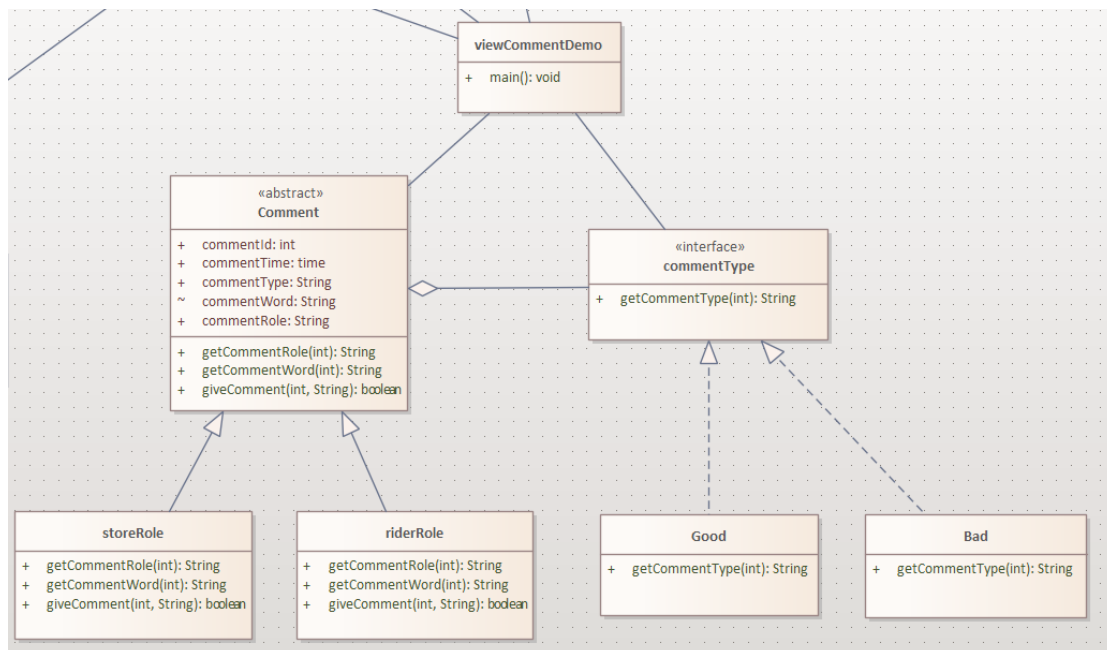
图表 6-1- 1 Store 单例模式优化

2) Order 单例模式优化



图表 6-1- 2 Order 单例模式优化

2. 桥接模式优化



图标 6-2- 1 Comment 类桥接模式优化

1) 设计思想

当一个类存在两个独立变化的维度，且这两个维度都需要进行扩展时，可以使用桥接模式

2) 设计问题探讨

Comment 类存在 commentRole 和 commentType 两个变化维度，可以扩展为“商家”“骑手”和“好评”“差评”。

3) 设计说明

viewControlDemo 类使用 Comment 类来查看与进行评价；Comment 类为抽象类，storeRole 类和 riderRole 类是抽象类的实例化；commentType 为接口，Good 和 Bad 是实现接口的实体类

3. Java 编程

1) Order 类

```
package 在线外卖点餐系统;

public class Order {
```



```

    public int orderId;
    public double orderPrice;
    public String orderTime;
    public String limitedTime;
    private String deliverStatus;
    private String foodStatus;
    public Discount m_Discount = new Discount();
    public Dish m_Dish=new Dish();
    public Comment m_Comment;
    public Consumer m_consumer=new Consumer();
    public Rider m_rider;
    public Store m_store;

    private static Order instance = new Order();

    public Order() {

    }

    public void finalize() throws Throwable {

    }

    // 饿汉单例模式
    public static Order getInstance() {
        return instance;
    }
    // 创建订单
    public void createOrder() {
        m_Dish.selectDish(m_Dish.dishName, m_Dish.dishNum);
        setOrderId();
        double price=setPrice(m_Dish);
    }

    public void setOrderId() {
        System.out.println("订单号为 1! ");
    }

    /**
     *
     * @param dish
     */

```

```

public double setPrice(Dish dish) {
    System.out.println("商品原价为 100 元! ");
    return dish.dishPrice*dish.dishNum;
}

/**
 *
 * @param price
 */
public boolean alterPrice(double price) {
    return true;
}

/**
 *
 * @param price
 */
public boolean payOrder(double price) {
    if(m_Discount.useDiscount()) {
        System.out.println("成功使用优惠券! ");
        if(alterPrice(price)) {
            System.out.println("实付 80 元");
        }
    }
    return false;
}

/**
 *
 * @param orderId
 */
public boolean acceptOrder(int orderId) {
    return true;
}

public boolean refuseOrder(int orderId) {
    return true;
}

/**
 *
 * @param orderId
 */
public void finishOrder(int orderId) {
    System.out.println("请求完成订单"+orderId);
}

```

```

    }

    public boolean addOrderTime() {
        return false;
    }

    /**
     *
     * @param limitedTime
     */
    public boolean checkTime(String limitedTime) {
        return true;
    }

    public void setDeliverStatus(boolean button) {
        if(!button) {
            System.out.println("骑手配送中……");
        }else {
            System.out.println("骑手完成订单");
        }
    }

    public void setDeliverTime() {
        System.out.println("----- 订 单 于 16 : 11pm 完 成
-----");
    }

    public void setFoodStatus(boolean button) {
        if(!button) {
            System.out.println("商家备餐中……");
        }else {
            System.out.println("商家已出餐");
        }
    }

    public void setFoodTime() {
        System.out.println("----- 备 餐 于 16 : 00pm 完 成
-----");
    }

    /**
     *
     * @param orderId
     */

```

```
        public Order viewOrder(int orderId) {  
            return null;  
        }  
    }  
} //end Order
```

2) Store 类

```
package 在线外卖点餐系统;  
  
public class Store extends User {  
  
    public String storeName;  
    public String storeAddress;  
    public String storeOpeningHours;  
    String hostName;  
    private Integer hostIdCard;  
    protected Integer storeOrderNum;  
    public Order m_Order;  
    public storeRole m_Comment=new storeRole();  
    public Dish m_Dish = new Dish();  
  
    private static Store instance = new Store();  
  
    public Store() {  
    }  
    public static Store getInstance() {  
        return instance;  
    }  
  
    public void finalize() throws Throwable {  
        super.finalize();  
    }  
    public boolean addOrderNum() {  
        return true;  
    }  
  
    /**  
     *  
     * @param name or dish  
     */  
    public Store searchStore(String keyword) {  
        System.out.println("找到商家 1");  
    }  
}
```

```
        m_Dish.SearchStore(keyword);  
        return null;  
    }  
} //end Store
```

3) viewCommentDemo 类

```
package 在线外卖点餐系统;  
  
public class viewCommentDemo {  
    viewCommentDemo() {  
  
    }  
    public static void main(String[] args) {  
        storeRole store=new storeRole();  
        riderRole rider=new riderRole();  
        System.out.println(store.getCommentRole(1));  
        System.out.println(rider.getCommentRole(1));  
    }  
}  
  
interface commentType{  
    public String getCommentType(int commendId);  
}  
  
class Good implements commentType{  
    public String getCommentType(int commendId) {  
        return "Good";  
    }  
}  
  
class Bad implements commentType{  
    public String getCommentType(int commendId) {  
        return "Bad";  
    }  
}  
  
abstract class Comment{  
  
    public int commentId;  
    public String commentTime;  
    public String commentType;
```

```

String commentWord;
public String commentRole;
public commentType m_commentType = new Good();
public commentType m_commentType1 = new Bad();

public Comment() {

}

public abstract String getCommentRole(int commendId);

public abstract String getCommentWord(int commendId);

public abstract boolean giveComment(int commendId, String s);
}

class storeRole extends Comment{

    public String getCommentRole(int commendId) {
        return "对商家"+"的评价类型为
"+m_commentType.getCommentType(1);
    }
    String s;
    public String getCommentWord(int commendId) {
        return s;
    }

    public boolean giveComment(int commendId, String s) {
        this.s= s;
        return true;
    }
}

class riderRole extends Comment{
    public String getCommentRole(int commendId) {
        return "对骑手"+"的评价类型为
"+m_commentType1.getCommentType(1);
    }
    String s;
    public String getCommentWord(int commendId) {
        return s;
    }
}

```

```

        public boolean giveComment(int commendId,String s) {
            this.s=s;
            return true;
        }
    }
}

```

运行结果截图为：

The screenshot shows an IDE with two panels. The top panel displays the source code for `viewCommentDemo.java`. The code defines an abstract class `Comment` with methods `getCommentWord` and `giveComment`, and a concrete class `storeRole` that extends `Comment` and implements these methods. The bottom panel shows the console output, which displays the results of the program's execution.

```

viewCommentDemo.java
47
48
49 public abstract String getCommentWord(int commendId);
50
51 public abstract boolean giveComment(int commendId,String s);
52
53 }
54
55 class storeRole extends Comment{
56
57     public String getCommentRole(int commendId) {
58         return "对商家"+"的评价类型为"+m_commentType.getCommentType(1);
59     }
60     String s;
61     public String getCommentWord(int commendId) {
62
63     }
64 }

```

```

Console
<terminated> viewCommentDemo [Java Application] D:\eclipse-java-2021-06-R-win32-x86_64\eclipse\plugins\
对商家的评价类型为Good
对骑手的评价类型为Bad

```

图表 6-3- 1 viewCommentDemo 运行结果

4) ConsumerDemo 类

```

package 在线外卖点餐系统;

import java.util.Scanner;

public class ConsumerDemo {
    public static void main(String[] args) {

        Consumer consumer1 = new Consumer();
        System.out.println("成功创建顾客 1! ");
        consumer1.m_Order=Order.getInstance();
        Store store1=Store.getInstance();
        Scanner sc= new Scanner(System.in);
        String keyword = sc.next();

        System.out.println("顾客 1 查找商家: ");
        store1.searchStore(keyword);
    }
}

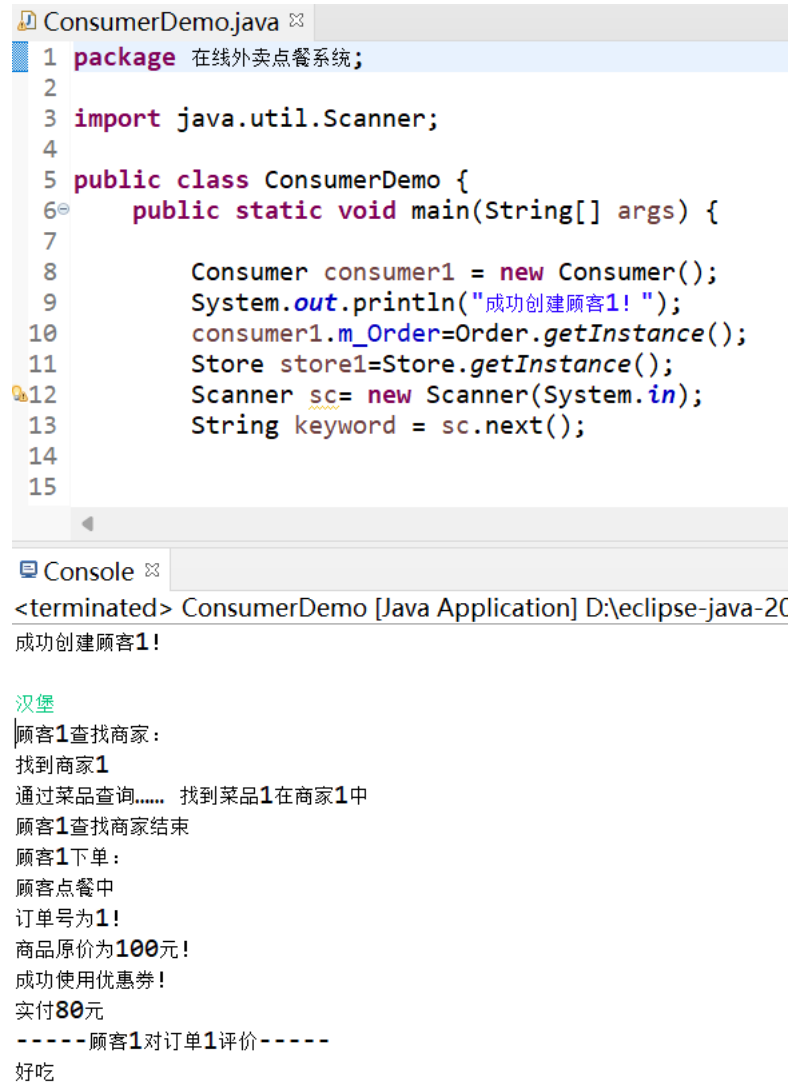
```



```
System.out.println("顾客 1 查找商家结束");
System.out.println("顾客 1 下单：");
consumer1.m_Order.createOrder();
consumer1.m_Order.payOrder(1);

System.out.println("-----顾客 1 对订单 1 评价-----");
boolean a=consumer1.m_Comment.giveComment(1, "好吃");
if(a) {
    System.out.println(consumer1.m_Comment.s);
}
}
```

运行结果截图为：



```
ConsumerDemo.java
1 package 在线外卖点餐系统;
2
3 import java.util.Scanner;
4
5 public class ConsumerDemo {
6     public static void main(String[] args) {
7
8         Consumer consumer1 = new Consumer();
9         System.out.println("成功创建顾客1!");
10        consumer1.m_Order=Order.getInstance();
11        Store store1=Store.getInstance();
12        Scanner sc= new Scanner(System.in);
13        String keyword = sc.next();
14
15    }
16 }

Console
<terminated> ConsumerDemo [Java Application] D:\eclipse-java-20
成功创建顾客1!

汉堡
顾客1查找商家:
找到商家1
通过菜品查询..... 找到菜品1在商家1中
顾客1查找商家结束
顾客1下单:
顾客点餐中
订单号为1!
商品原价为100元!
成功使用优惠券!
实付80元
-----顾客1对订单1评价-----
好吃
```

图表 6-3- 2 ConsumerDemo 运行结果

5) RiderDemo 类

```
package 在线外卖点餐系统;

public class RiderDemo {
    public static void main(String[] args) {
        Rider rider1 = new Rider();
        rider1.m_Order = Order.getInstance();
        if(rider1.m_Order.acceptOrder(1)) {
            System.out.println("骑手接收订单 1!");
        }
        rider1.m_Order.setDeliverStatus(false);
        rider1.m_Order.finishOrder(1);
    }
}
```

```

        if(!rider1.m_Order.m_consumer.checkAddress(rider1.m_Order.
m_consumer.consumerAddress)) {
            System.out.println("骑手未送达指定地点");
        }else if(!rider1.m_Order.checkTime("30")) {
            System.out.println("骑手已超时");
            rider1.m_Order.setDeliverStatus(true);
            rider1.m_Order.setDeliverTime();
        }else {
            System.out.println("骑手在规定时间内送达指定地点");
            rider1.m_Order.setDeliverStatus(true);
            rider1.m_Order.setDeliverTime();
        }
        if(rider1.addOrderNum()) {
            System.out.println("骑手完成订单加一");
        }

        viewCommentDemo vcd = new viewCommentDemo();
        System.out.println("得到的评价如下：");
        vcd.main(null);
    }
}

```

运行结果截图为：

```

RiderDemo.java
1 package 在线外卖点餐系统;
2
3 public class RiderDemo {
4     public static void main(String[] args) {
5         Rider rider1 = new Rider();
6         rider1.m_Order = Order.getInstance();
7         if(rider1.m_Order.acceptOrder(1)) {
8             System.out.println("骑手接收订单1!");
9         }
10        rider1.m_Order.setDeliverStatus(false);
11        rider1.m_Order.finishOrder(1);
12        if(!rider1.m_Order.m_consumer.checkAddress(rider1.m_Order.m_consumer.consumerAddress)) {
13            System.out.println("骑手未送达指定地点");
14        }else if(!rider1.m_Order.checkTime("30")) {
15            System.out.println("骑手已超时");
16        }
17    }
18 }

```

```

<terminated> RiderDemo [Java Application] D:\eclipse-java-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\java.exe
成功创建骑手1!
骑手接收订单1!
骑手配送中.....
请求完成订单1
骑手在规定时间内送达指定地点
骑手完成订单
-----订单于 16:11pm 完成-----
骑手完成订单加一
得到的评价如下:
对商家的评价类型为Good
对骑手的评价类型为Bad

```

图表 6-3- 3 RiderDemo 运行结果

6) StoreDemo 类

```
package 在线外卖点餐系统;

public class StoreDemo {
    public static void main(String[] args) {

        Store store=Store.getInstance();
        store.m_Order = Order.getInstance();

        boolean button = true;
        if(button) {
            if(store.m_Order.acceptOrder(1)) {
                System.out.println("商家接受订单 1! ");
                store.m_Order.setFoodStatus(false);
            }
        }else {
            if(store.m_Order.refuseOrder(1)) {
                System.out.println("商家拒绝订单 1! ");
                return;
            }
        }
        store.m_Order.finishOrder(1);
        store.m_Order.setFoodStatus(true);
        store.m_Order.setFoodTime();
        if(store.addOrderNum()) {
            System.out.println("商家完成订单加一");
        }

        System.out.println("得到的评价如下: ");
        viewCommentDemo vcd = new viewCommentDemo();
        vcd.main(null);

    }
}
```

运行结果截图为：

```
StoreDemo.java
1 package 在线外卖点餐系统;
2
3 public class StoreDemo {
4     public static void main(String[] args) {
5
6         Store store=Store.getInstance();
7         store.m_Order = Order.getInstance();
8
9
10        boolean button = true;
11        if(button) {
12            if(store.m_Order.acceptOrder(1)) {
13                System.out.println("商家接受订单1!");
14                store.m_Order.setFoodStatus(false);
15            }
16        }
17    }
18 }
```

```
Console
<terminated> StoreDemo [Java Application] D:\eclipse-java-2021-06-R-
商家接受订单1!
商家备餐中.....
请求完成订单1
商家已出餐
-----备餐于 16:00pm 完成-----
商家完成订单加一
得到的评价如下:
对商家的评价类型为Good
对骑手的评价类型为Bad
```

图表 6-3- 4 StoreDemo 运行结果