PA1459/PA1460 Example: BurgerOrderer

Mikael Svahnberg

February 10, 2021

Contents

1 Use Case Order Food

Use Case Order food

Actors Customer

Description A customer arrives at the BurgerOrderer, selects a meal, configures their burger, and orders it.

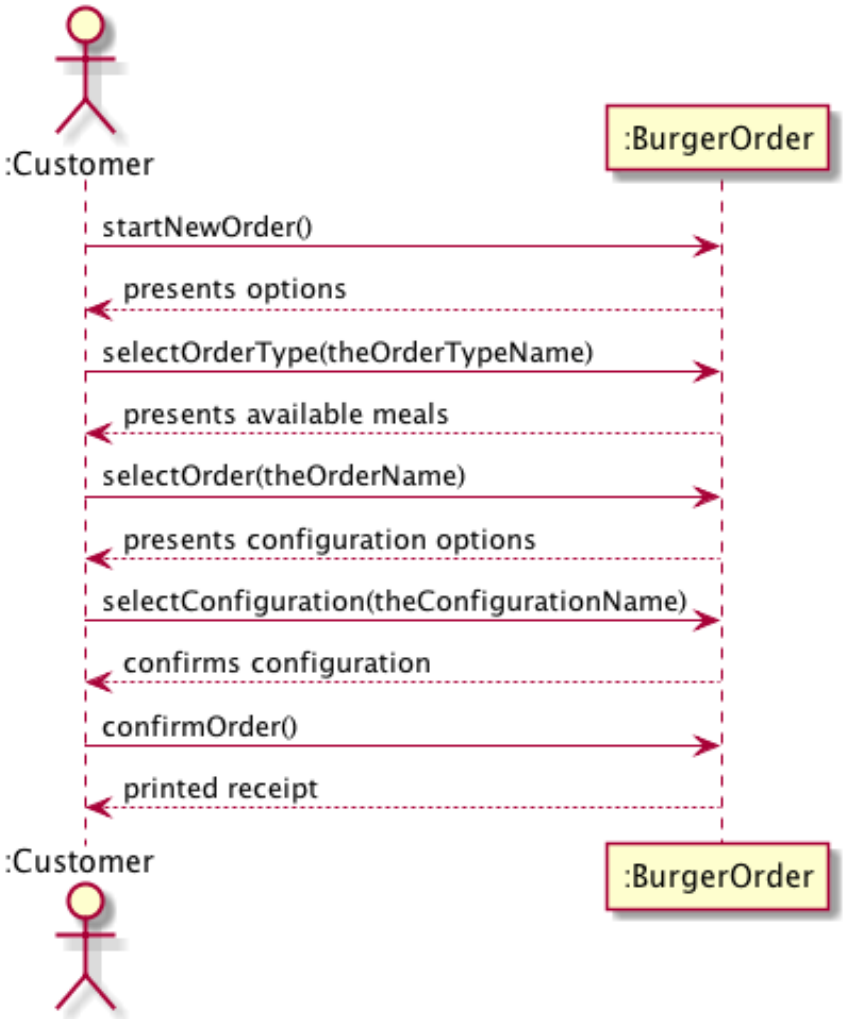
Related Use Cases Pay for order

Main course of events

Actor	System
1. Customer arrives at BurgerOrderer and starts a new order.	
	2. System presents options [single burger, meal, dessert, dring]
3. Customer selects "meal"	
	4. System presents available meals
5. Customer selects a specific meal.	
	6. System adds the selected meal to the order. 7. System presents configuration options
8. customer selects "no onions"	
	9. System adds "no onions" to order.
10. customer selects "more bacon!"	
	11. System adds "more bacon!" to order.
12. Customer confirms order.	
	13. System initiates use case pay for order 14. System places order to kitchen and prints receipt.

2 System Sequence Diagram

```
actor ":Customer" as cus
participant ":BurgerOrder" as sys

cus -> sys : startNewOrder()
sys --> cus : presents options

cus -> sys : selectOrderType(theOrderTypeName)
sys --> cus : presents available meals

cus -> sys : selectOrder(theOrderName)
sys --> cus : presents configuration options

cus -> sys : selectConfiguration(theConfigurationName)
sys --> cus : confirms configuration

cus -> sys : confirmOrder()
sys --> cus : printed receipt
```

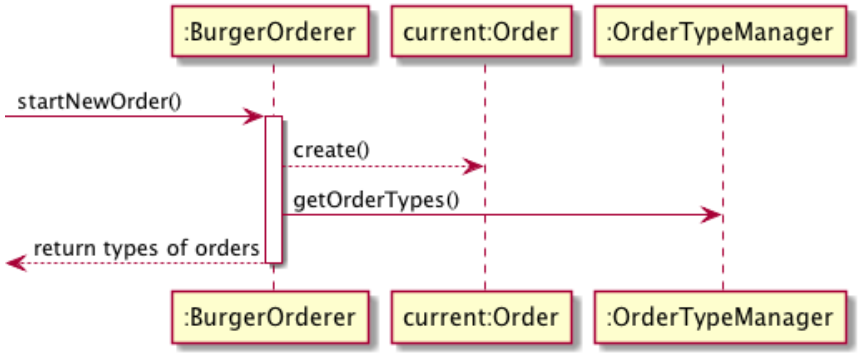


3 Interaction Diagrams (Sequence Diagrams)

3.1 startNewOrder()

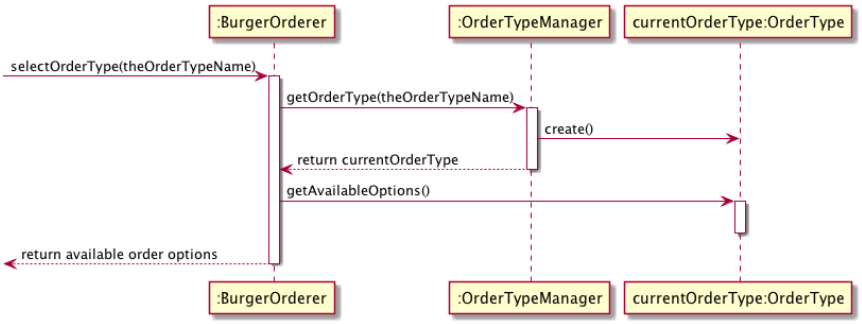
```
participant ":BurgerOrderer" as sys

[-> sys : startNewOrder()
activate sys

sys --> "current:Order"  : create()
```

```
sys -> ":OrderTypeManager" : getOrderTypes()

[<-- sys : return types of orders
deactivate sys
```



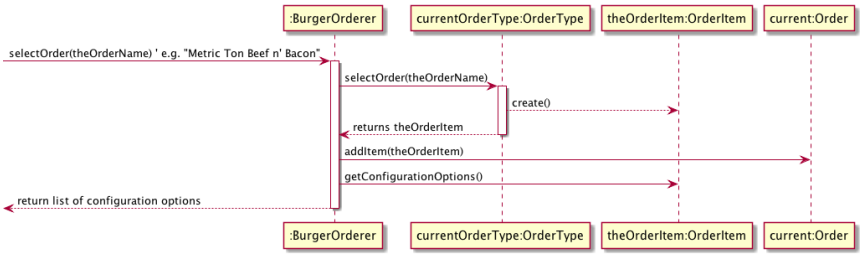
3.2 selectOrderType()

```
participant ":BurgerOrderer" as sys

[-> sys : selectOrderType(theOrderTypeName)
activate sys
sys -> ":OrderTypeManager" : getOrderType(theOrderTypeName)
activate ":OrderTypeManager"
":OrderTypeManager" -> "currentOrderType:OrderType" : create()
":OrderTypeManager" --> sys : return currentOrderType
deactivate ":OrderTypeManager"

sys -> "currentOrderType:OrderType" : getAvailableOptions()
activate "currentOrderType:OrderType"
deactivate "currentOrderType:OrderType"

[<-- sys : return available order options
deactivate sys
```



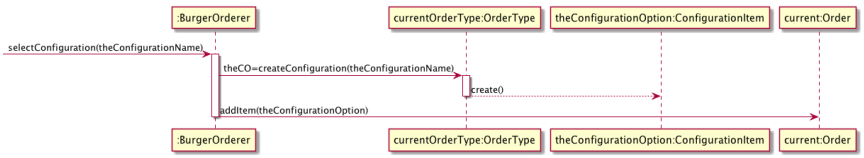
3.3 selectOrder()

```
participant ":BurgerOrderer" as sys

[-> sys : selectOrder(theOrderName) ' e.g. "Metric Ton Beef n' Bacon"
activate sys
sys -> "currentOrderType:OrderType" : selectOrder(theOrderName)
activate "currentOrderType:OrderType"

"currentOrderType:OrderType" --> "theOrderItem:OrderItem" : create()

"currentOrderType:OrderType" --> sys : returns theOrderItem
deactivate "currentOrderType:OrderType"

sys -> "current:Order" : addItem(theOrderItem)
sys -> "theOrderItem:OrderItem" : getConfigurationOptions()

[<-- sys : return list of configuration options
deactivate sys
```



3.4 selectConfiguration()

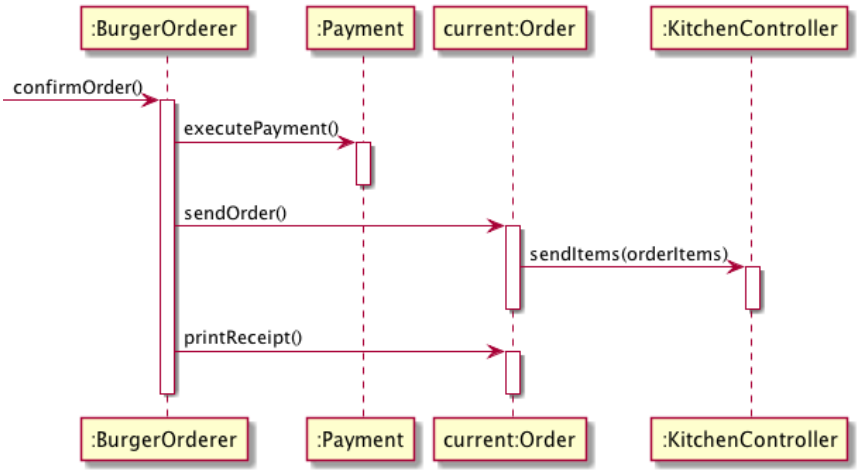
```
participant ":BurgerOrderer" as sys

[-> sys : selectConfiguration(theConfigurationName)
' e.g. "more bacon!"
activate sys
sys -> "currentOrderType:OrderType" : theCO=createConfiguration(theConfigurationName)
activate "currentOrderType:OrderType"
"currentOrderType:OrderType" --> "theConfigurationOption:ConfigurationItem" : create()
deactivate "currentOrderType:OrderType"

sys -> "current:Order" : addItem(theConfigurationOption)

deactivate sys
```



3.5 confirmOrder()

```
participant ":BurgerOrderer" as sys

[-> sys : confirmOrder()
activate sys

sys -> ":Payment" : executePayment()
activate ":Payment"
deactivate ":Payment"

sys -> "current:Order" : sendOrder()
activate "current:Order"
"current:Order" -> ":KitchenController" : sendItems(orderItems)
activate ":KitchenController"
deactivate ":KitchenController"
deactivate "current:Order"

sys -> "current:Order" : printReceipt()
```

```
activate "current:Order"
deactivate "current:Order"
deactivate sys
```



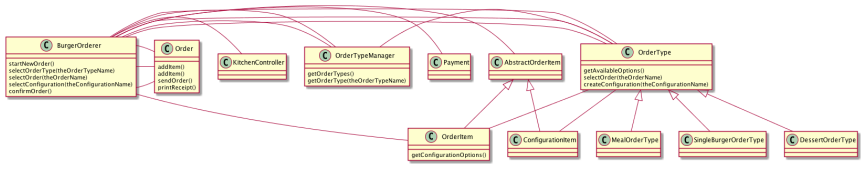
4 Class Diagram – First version

In this version, I have simply merged all of the interaction diagrams above. As is seen, this means that associations between classes are duplicated, and some associations are made to the sub-class when they should be moved up to a super-class instead. I present this as a first version, and then I will clean it up and simplify it a bit.

```
' startNewOrder()
' --------------------
class BurgerOrderer
class Order
class OrderTypeManager

BurgerOrderer : startNewOrder()
OrderTypeManager : getOrderTypes()

BurgerOrderer - Order
BurgerOrderer - OrderTypeManager

' selectOrderType()
```

```
' --------------------
class BurgerOrderer
class OrderTypeManager
class OrderType

BurgerOrderer : selectOrderType(theOrderTypeName)
OrderTypeManager : getOrderType(theOrderTypeName)
OrderType : getAvailableOptions()

BurgerOrderer - OrderTypeManager
BurgerOrderer - OrderType
OrderTypeManager - OrderType

' selectOrder()
' --------------------
class BurgerOrderer
class OrderType
class OrderItem
class Order

BurgerOrderer : selectOrder(theOrderName)
OrderType : selectOrder(theOrderName)
Order : addItem()
OrderItem : getConfigurationOptions()

BurgerOrderer - OrderType
OrderType - OrderItem
BurgerOrderer - OrderItem
BurgerOrderer - Order

' selectConfiguration()
' --------------------
class BurgerOrderer
class OrderType
class ConfigurationItem
class Order

BurgerOrderer : selectConfiguration(theConfigurationName)
OrderType : createConfiguration(theConfigurationName)
Order : addItem()
```

```
BurgerOrderer - OrderType
OrderType - ConfigurationItem
BurgerOrderer - AbstractOrderItem

' adding a few inheritance hierarchies that I think will be needed
AbstractOrderItem <|-- ConfigurationItem
AbstractOrderItem <|-- OrderItem

OrderType <|-- MealOrderType
OrderType <|-- SingleBurgerOrderType
OrderType <|-- DessertOrderType


' confirmOrder()
' --------------------
class BurgerOrderer
class Payment
class Order
class KitchenController

BurgerOrderer : confirmOrder()
Order : sendOrder()
Order : printReceipt()

BurgerOrderer - Payment
BurgerOrderer - Order
BurgerOrderer - KitchenController
```



5 Class Diagram – Simplified

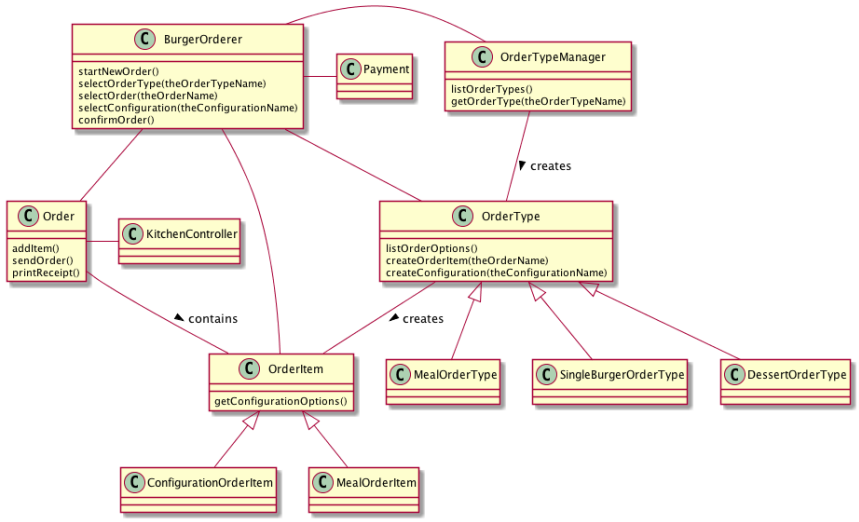
Please see the comments in the code below for information about what I have done and why,.

```
' startNewOrder()
' --------------------
class BurgerOrderer
class Order
class OrderTypeManager

BurgerOrderer : startNewOrder()

' Replaced "getOrderTypes()"  with "listOrderTypes()" since this is slightly clearer.
OrderTypeManager : listOrderTypes()

' Replaced the single dash with a double dash to put BurgerOrderer on top of the other
BurgerOrderer -- Order

BurgerOrderer - OrderTypeManager

' selectOrderType()
' --------------------
' I don't really need to re-declare BurgerOrderer or OrderTypeManager
' but nothing is added to the final result if I keep them so for simplicity's
' sake, I'll leave them as they are.
class BurgerOrderer
class OrderTypeManager

' For reasons that I will expand upon later
' I want OrderType to be abstract.
abstract class OrderType

BurgerOrderer : selectOrderType(theOrderTypeName)
OrderTypeManager : getOrderType(theOrderTypeName)

' Replaced "getAvailableOptions()" with "listOrderOptions()"
OrderType : listOrderOptions()

' Remove this association to avoid multiple lines in the diagram
```

```
' BurgerOrderer - OrderTypeManager

' Replaced single dash with double dashes
BurgerOrderer -- OrderType
OrderTypeManager -- OrderType : creates >

' selectOrder()
' --------------------
class BurgerOrderer
class OrderType
class OrderItem
class Order

BurgerOrderer : selectOrder(theOrderName)

' renamed selectOrder() => createOrderItem()
OrderType : createOrderItem(theOrderName)

Order : addItem()
OrderItem : getConfigurationOptions()


' Duplicates
'BurgerOrderer - OrderType
'BurgerOrderer - Order

' Replaced single dash with double dashes
' Added information that OrderType merely creates OrderItem
OrderType -- OrderItem : creates >
BurgerOrderer -- OrderItem

' selectConfiguration()
' --------------------
class BurgerOrderer
class OrderType
class Order

' See discussion below why I remove this
' class ConfigurationItem
```

```
BurgerOrderer : selectConfiguration(theConfigurationName)
OrderType : createConfiguration(theConfigurationName)

' Duplicate
' Order : addItem()

' Duplicates
'BurgerOrderer - OrderType

' The following two associations are a bit tricky. I want to abstract
' "ConfigurationItem" and "OrderItem" to something more generic, and I
' want to collectively call these OrderItems, i.e. the base class should
' be called OrderItem. With sub-classes ConfigurationOrderItem and
' -- perhaps -- MealOrderItem?  so the association from OrderType will go
' to the abstract base class OrderItem (even if it is a configurationOrderItem
' that is being created right now. And that makes the associations
'  duplicates to already stated associations above. So I remove them.

' OrderType - ConfigurationItem
' BurgerOrderer - AbstractOrderItem

' adding a few inheritance hierarchies that I think will be needed
' Renaming the OrderItem hierarchy as per the discussion above.
OrderItem <|-- ConfigurationOrderItem
OrderItem <|-- MealOrderItem

OrderType <|-- MealOrderType
OrderType <|-- SingleBurgerOrderType
OrderType <|-- DessertOrderType


' confirmOrder()
' --------------------
class BurgerOrderer
class Payment
class Order
class KitchenController

BurgerOrderer : confirmOrder()
```

```
Order : sendOrder()
Order : printReceipt()

BurgerOrderer - Payment

' Replace BurgerOrderer with Order since I mis-read the interaction diagram before
Order - KitchenController

' Duplicate
'BurgerOrderer - Order


' Add an association
Order - OrderItem : contains >
```



And there you have it. With this diagram we can now take a step back and look at a few things.

- First, `BurgerOrderer` is connected to everything! Is there anything we can do to avoid this?
- Second, the `OrderItem` inheritance hierarchy does not have that many methods currently. This *could* be because we have only modelled a single use case. But it can also indicate that maybe we do not need to have an inheritance hierarchy here. Maybe `OrderItem` with a few attributes can be sufficient.

- Third and likewise, the `OrderType` hierarchy is also suspiciously empty of methods.