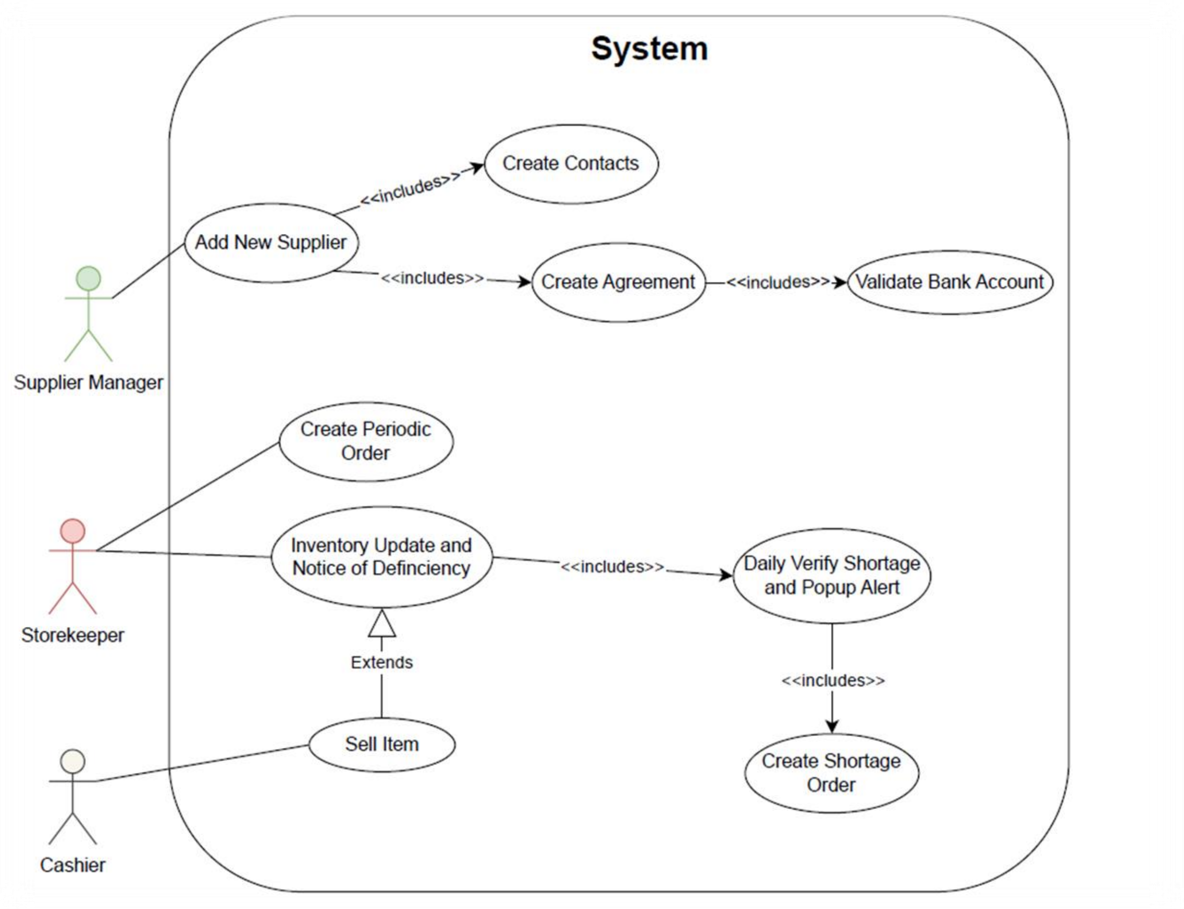
אפיון מערכת – מודול ספקים-מלאי

מגישים: גיא ביטון (207577875), גל חליפה (316262914), דן ויצמן (318940913), איתמר בראמי (207215203)

Use Case Diagram



Use Case Description:

Use case e
name: Periodic order from supplier

Textual Description: Create an order in fixed days from specific supplier.
List of Actors: Storekeeper
Pre-conditions:

a. There are at least one or more suppliers in our system.
b. Product IDs and quantities are chosen from the chosen supplier's agreement.
c. Day for the periodic order has been chosen (the periodic day that the order will be created every week).

Post-conditions:

a. One supplier is chosen for the order.
b. Periodic order has been created.
c. The products and their quantity have been successfully added to the order.
d. The system creates the order every week in the correct day.
e. The supplier's inventory has been updated.
f. The order history of the branch has been recorded.
g. The order history of each supplier has been recorded.

Main success scenario:

a. The storekeeper begins a new periodic order.
b. The storekeeper choose supplier that he wishes to order from.
c. The storekeeper chooses product IDs from the supplier agreement and their amount.
d. The storekeeper chooses periodic day for the order.
e. The storekeeper sends the order with the details to the system.
f. The system creates a periodic order from the chosen supplier.
g. Once we reach the order day, an order is created, the system sends a message to the storekeeper informing them of the estimated arrival date and confirmation that the order has been created.
h. The system updates for each product in the order the amount that ordered.
i. The system updates the supplier's inventory.
j. The system records the order in the branch order history.
k. The system records the order in the supplier's order history.
l. Payment is made to the supplier for any order.

Alternatives/Extensions:

a. At any time, System fails: Turning the system off and on, we will make sure that all processes that have started have already been saved.
b. The storekeeper inserts invalid supplier ID – The system sends a message to the storekeeper that there is not such supplier ID exists in the system and ask him to choose new supplier ID.
c. The storekeeper inserts product IDs that are not available for the chosen supplier - the system sends a message to the storekeeper that the product IDs are not available for this supplier in the system and ask him to choose new IDs.
d. The storekeeper inserts duplicate product IDs to the order - the system sends a message to the storekeeper that there is duplicated product IDs in the order and ask him to choose new IDs.
e. The storekeeper inserts negative value of products quantity - the system sends a message to the storekeeper that there are incorrect quantity values.
f. One or more products are no longer provided from the chosen supplier for the periodic order – the order will be created in the next week without those products.

g. One or more products are currently out of stock (not enough quantity in their inventory) from the chosen supplier for the periodic order – the order will be created in the current week without those products.

Use case f

Name: Order from supplier due to shortage.

Textual Description: Order from supplier due to shortage. This order is created due to the lack of one or more products in the branch and the need to replenish the branch.

List of Actors: Storekeeper

Pre-conditions:

a. The storekeeper executes daily verify shortage and gets popup alert about products that their quantity goes under the minimum.
b. The system create order that contains enough to cover the minimum amount of each product in the order.
c. There are at least one or more suppliers in our system.
d. The suppliers (as a group of suppliers) have in their inventory enough to cover the shortages received in the order of each product.

Post-conditions:

a. The chosen suppliers are those who able to supply the products at the shortest time.
b. One or more orders have been created.
c. The products and their quantity that the storekeeper ordered has been successfully added to the orders.
d. The supplier's inventory has been updated.
e. The order history of the branch has been recorded.
f. The order history of each supplier has been recorded.

Main success scenario:

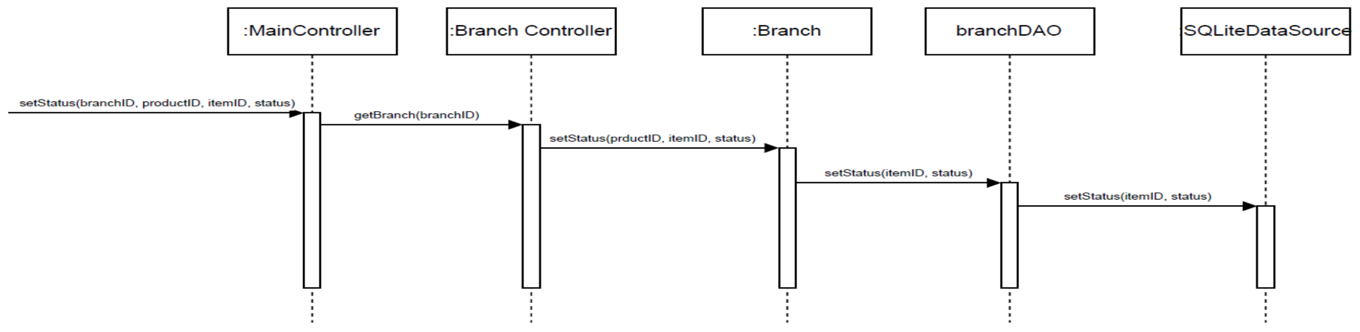
a. The storekeeper updates the inventory and gets pop up alert about products that goes under the minimum quantity.
b. The System begins a new shortage order.
c. The System adds the product IDs and the quantity of each product that goes under the minimum.
d. The system searches for a supplier that offers all products at the fastest time delivery, if isn't one the system searches for multiple suppliers that offer all products at the fastest time delivery.
e. The system creates an order from the chosen suppliers.
f. The system updates for each product in the order the amount that ordered.
g. The system updates the supplier's inventory.
h. The system records the order in the branch order history.
i. The system records the order in the supplier's order history.
j. Once the order has been created, the system sends a message to the storekeeper informing them of the estimated arrival date and confirmation that the order has been created.
k. Payment is made to the suppliers.

Alternatives/Extensions:

a. At any time, System fails: Turning the system off and on, we will make sure that all processes that have started have already been saved.
b. There is not one supplier or multiple suppliers (as a group of suppliers) that have in their inventory enough to cover the shortages received in the order of each product - The system will send response that the order can't be created.

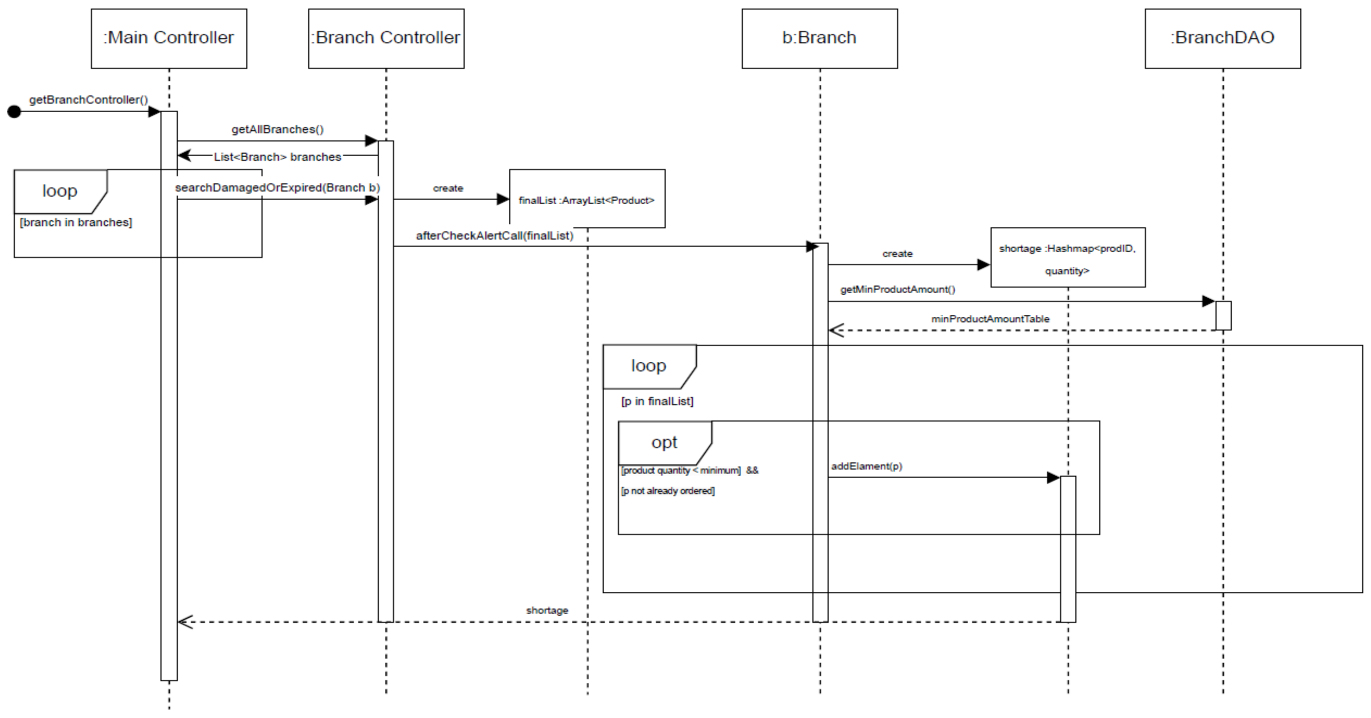
<u>Contracts + Sequence Diagram:</u>
<u>Use Case f – Order due to shortage</u>
Contract CO1: Update Inventory

Operation: Cross	setStatus(int branchID, int itemID)
References:	Use Cases: Create Shortage Order
Preconditions:	There is an item that sold/expired/damaged.
Postconditions:	- item.status has changed (attribute modification).


:MainController
:Branch Controller
:Branch
branchDAO
SQLiteDataSource
setStatus(branchID, productID, itemID, status)
getBranch(branchID)
setStatus(prductID, itemID, status)
setStatus(itemID, status)
setStatus(itemID, status)

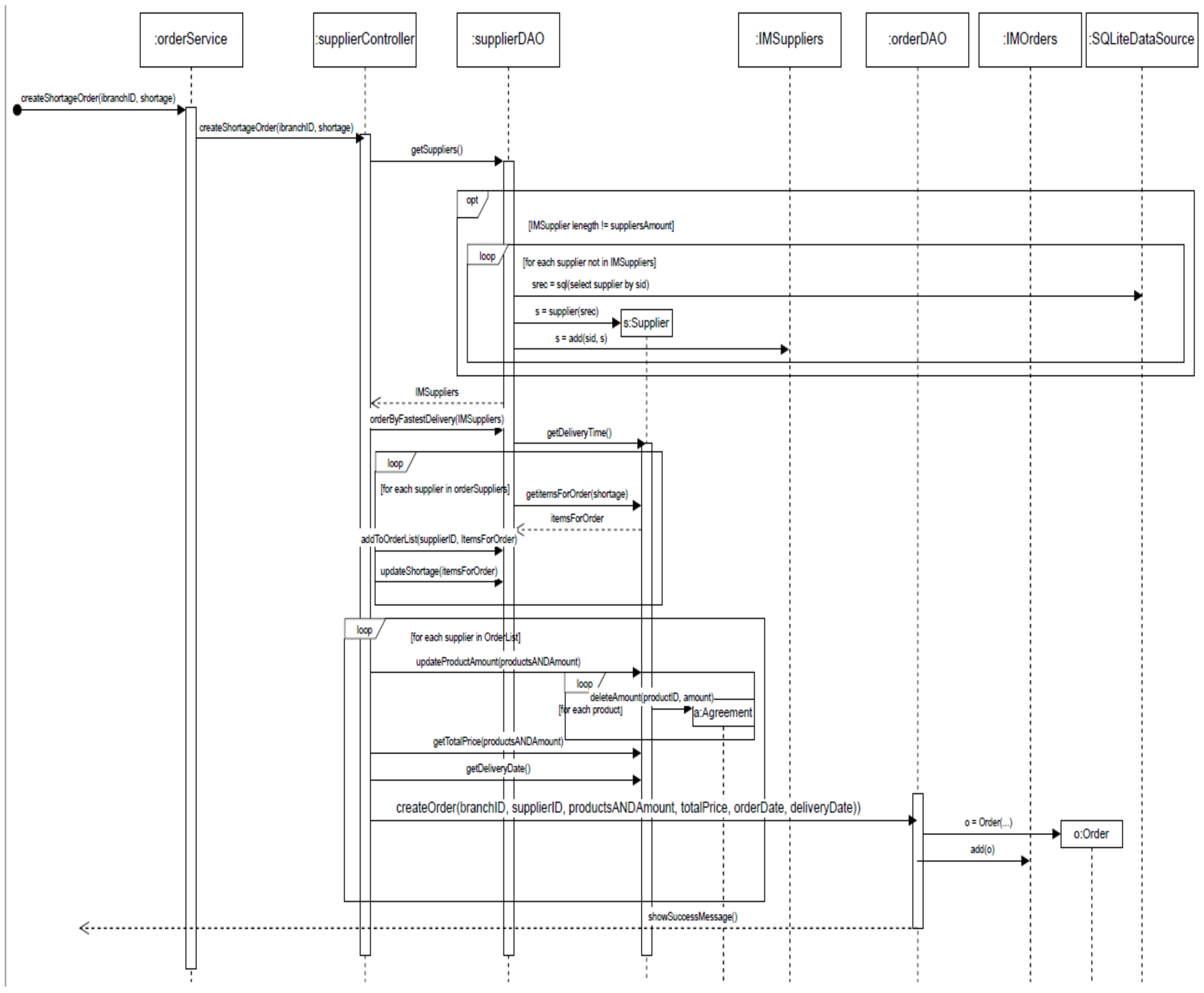
Contract CO2: Minimum Quantity Alert

Operation: Cross	afterCheckAlertCall(ArrayList<Product> productArray)
References:	Use Cases: Create Shortage Order
Preconditions:	- There are products that their quantity goes under the minimum. - The products that goes under the minimum do not order yet.
Postconditions:	- The corresponded product ID in which the amount is under the minimum been added as a key to the shortage products hash (instance creation). - The amount that needs to be order has been assigned to the value of the hash (attribute modification).



Contract CO3: createOrder

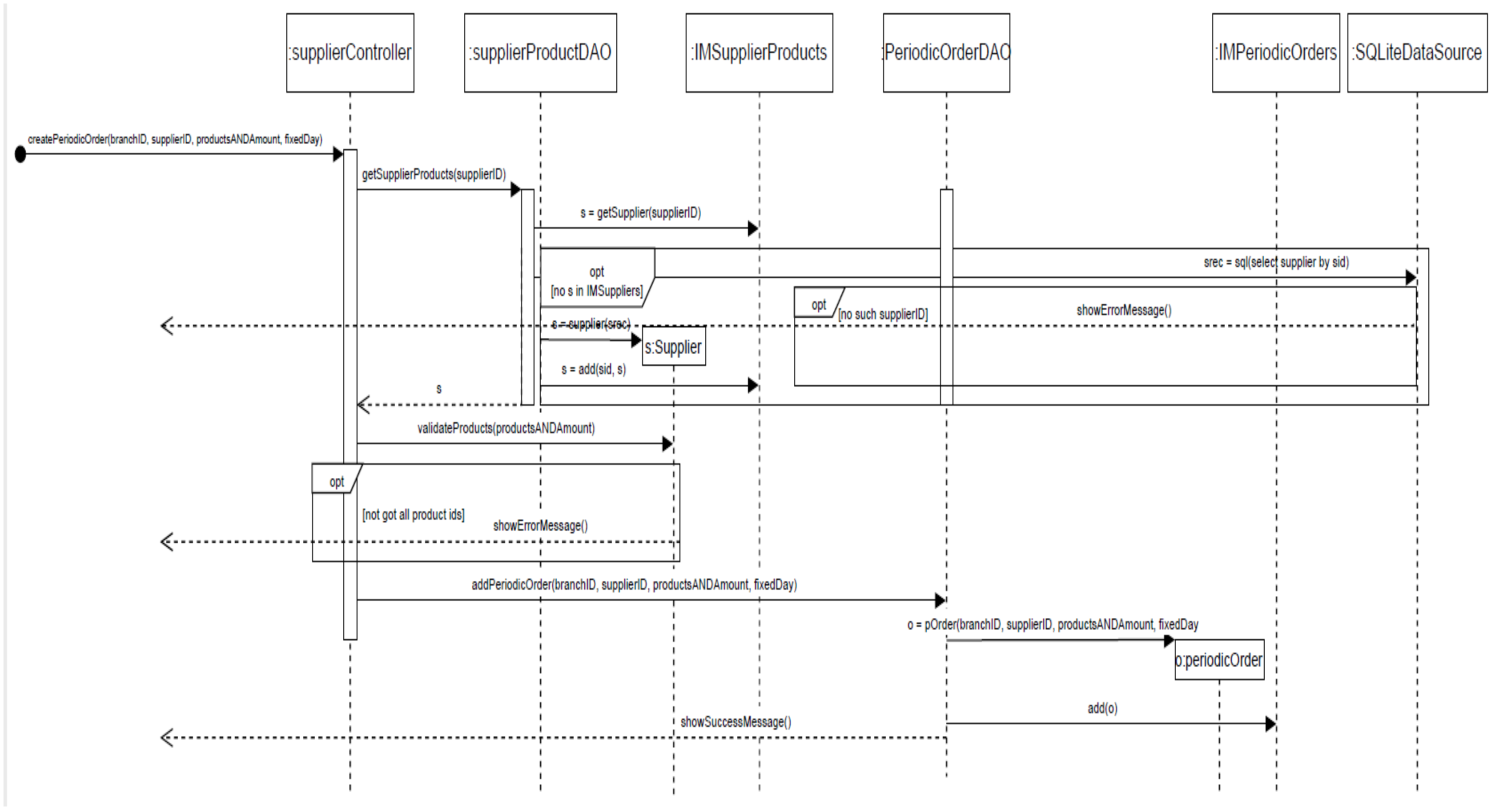
Operation: Cross	createShortageOrder(int branchID, Hashmap<productid: Integer, quantity: Integer> shortage)
References:	Use Cases: Create Shortage Order
Preconditions:	There are products that their quantity goes under the minimum.
Postconditions:	- An ArrayList instance orders was created (instance creation). - each order in orders was associated with Supplier (association formed); (To add it to the historical order records of the current supplier). - each order in orders was associated with the Branch (association formed); (To add it to the historical order records of the current branch). - Attributes of each order were initialized (attribute modification).


:orderService
:supplierController
:supplierDAO
:IMSuppliers
:orderDAO
:IMOrders
:SQLiteDataSource
createShortageOrder(ibranchID, shortage)
createShortageOrder(ibranchID, shortage)
getSuppliers()
opt
[IMSupplier lenegth != suppliersAmount]
loop
[for each supplier not in IMSuppliers]
srec = sql(select supplier by sid)
s = supplier(srec)
s:Supplier
s = add(sid, s)
IMSuppliers
orderByFastestDelivery(IMSuppliers)
getDeliveryTime()
loop
[for each supplier in orderSuppliers]
getitemsForOrder(shortage)
itemsForOrder
addToOrderList(supplierID, ItemsForOrder)
updateShortage(itemsForOrder)
loop
[for each supplier in OrderList]
updateProductAmount(productsANDAmount)
loop
deleteAmount(productID, amount)
[for each product]
a:Agreement
getTotalPrice(productsANDAmount)
getDeliveryDate()
createOrder(branchID, supplierID, productsANDAmount, totalPrice, orderDate, deliveryDate))
o = Order(...)
o:Order
add(o)
showSuccessMessage()

Use Case e – Periodic Order

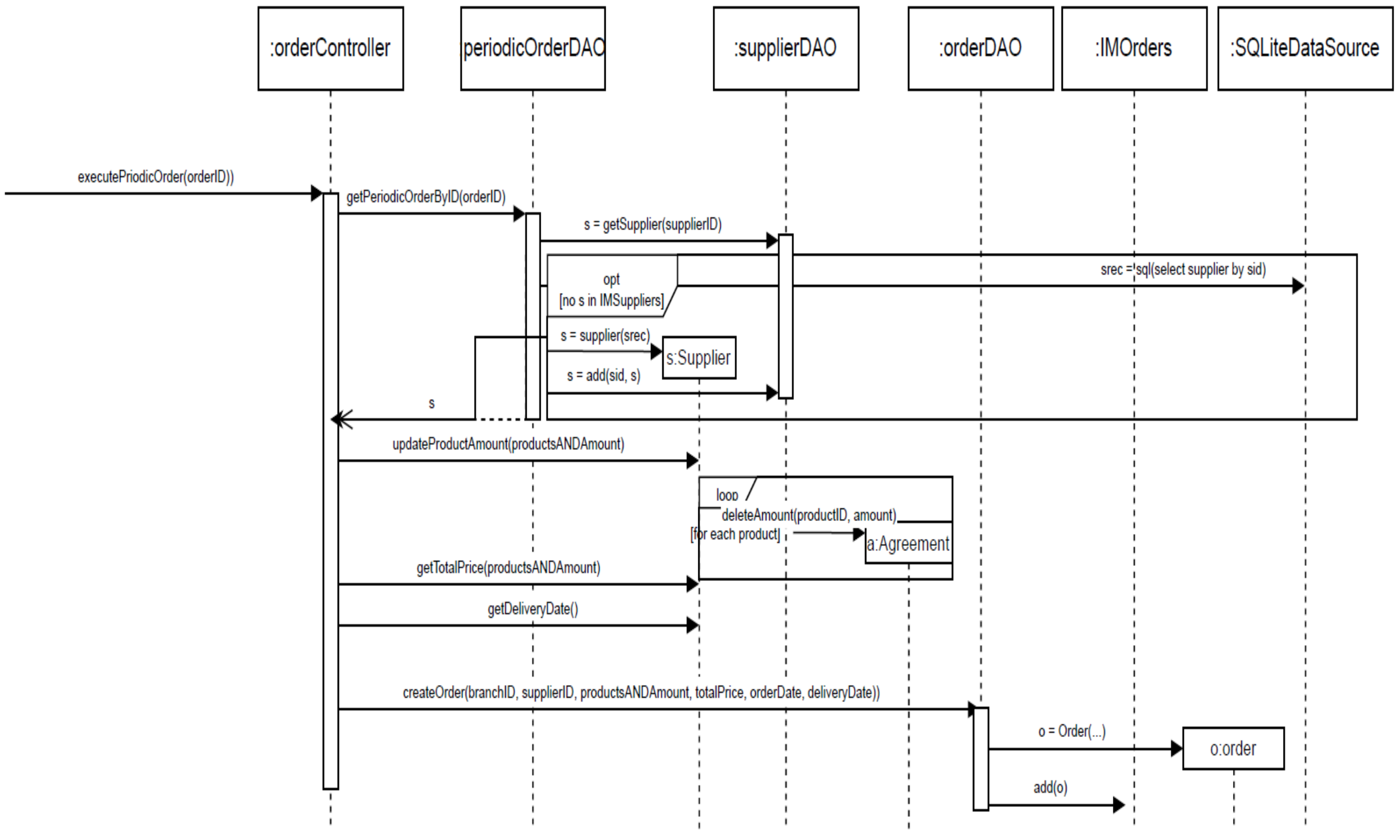
Contract CO1: createPeriodicOrder

Operation: Cross	createPeriodicOrder(int branchID, int supplierID, Hashmap<productid: Integer, quantity: Integer> products, LocalDate day)
References:	Use Cases: Create Periodic Order
Preconditions:	- There are at least one or more suppliers in our system. - Product IDs and quantities are exists in the chosen supplier's agreement.
Postconditions:	- A periodicOrder instance pOrder None was created (instance creation). - pOrder was associated with Supplier (association formed); - pOrder was associated with the Branch (association formed); - pOrder.day became day (attribute modification).

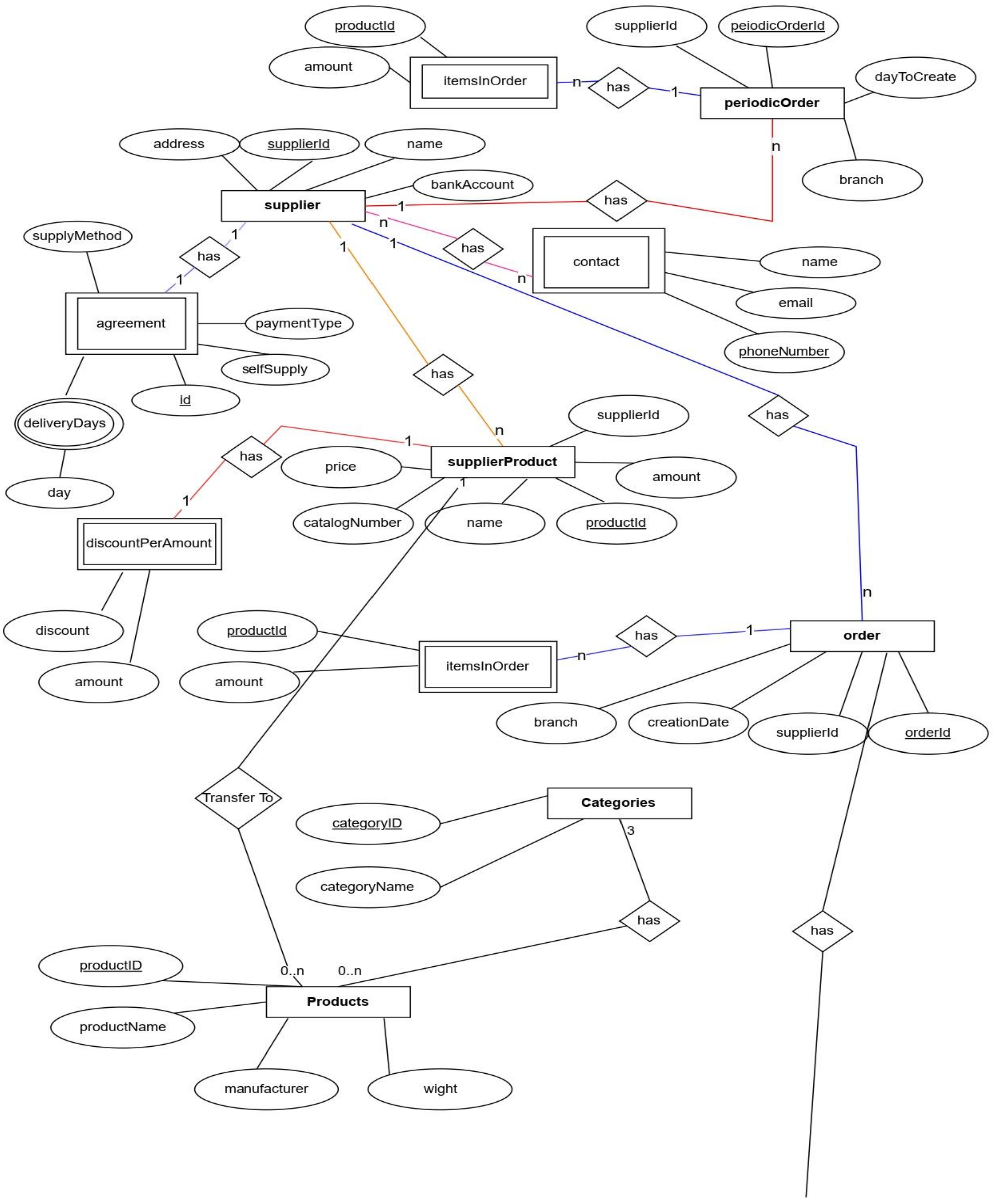

:supplierController
:supplierProductDAO
:IMSupplierProducts
:PeriodicOrderDAO
:IMPeriodicOrders
:SQLiteDataSource
createPeriodicOrder(branchID, supplierID, productsANDAmount, fixedDay)
getSupplierProducts(supplierID)
s = getSupplier(supplierID)
opt
[no s in IMSuppliers]
srec = sql(select supplier by sid)
opt
[no such supplierID]
showErrorMessage()
s = supplier(srec)
s:Supplier
s = add(sid, s)
s
validateProducts(productsANDAmount)
opt
[not got all product ids]
showErrorMessage()
addPeriodicOrder(branchID, supplierID, productsANDAmount, fixedDay)
o = pOrder(branchID, supplierID, productsANDAmount, fixedDay
o:periodicOrder
add(o)
showSuccessMessage()

Contract CO2: executePriodicOrder

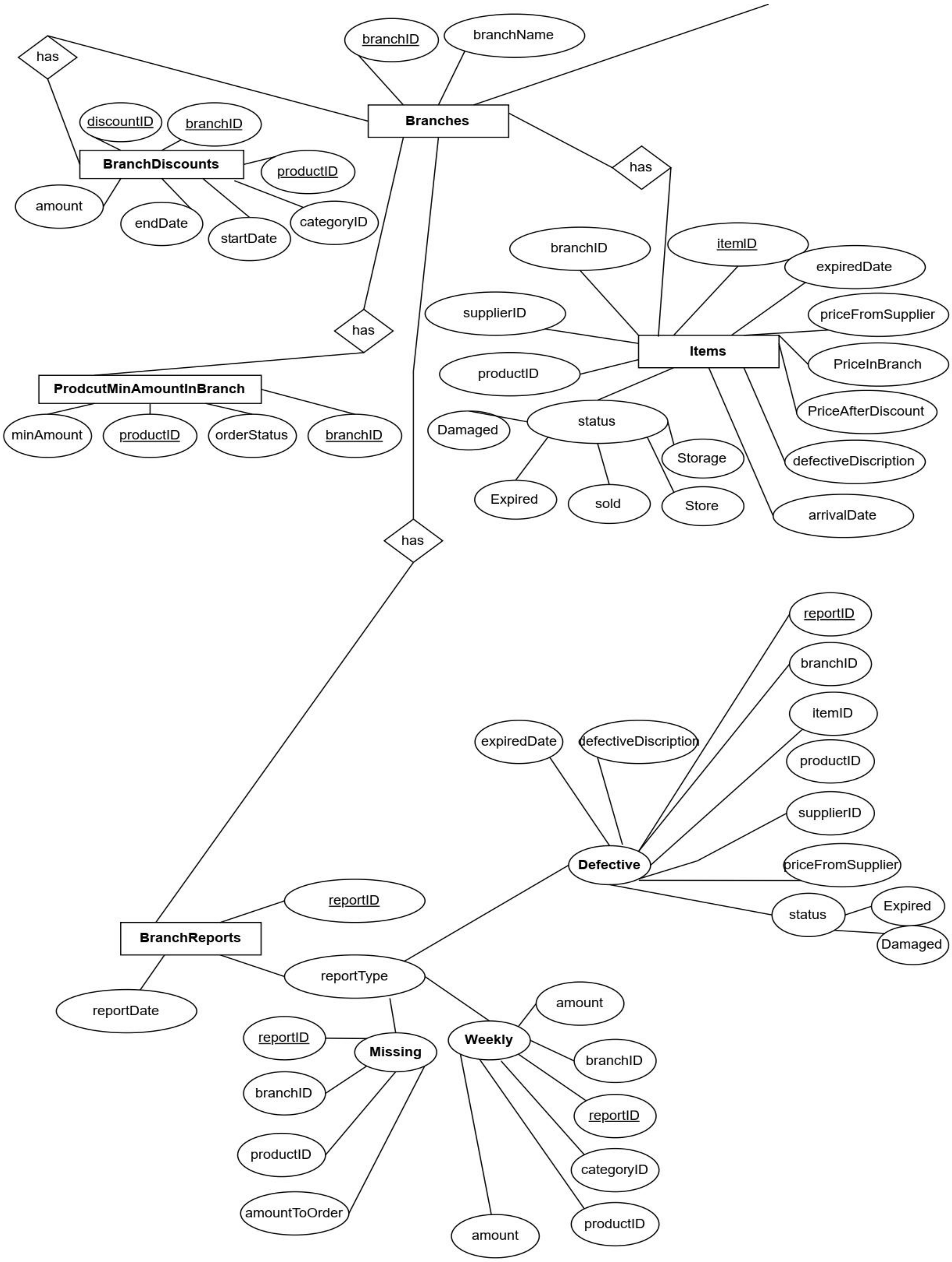
Operation: Cross	executePriodicOrder (int supplierID, int branchID, Hashmap<productid: Integer, quantity: Integer> shortage)
References:	Use Cases: Create Shortage Order
Preconditions:	The day of the periodic order has arrived**.**
Postconditions:	- An Hashmap<int, Product> instance pio was created (instance creation). - An Order instance order was created (instance creation). - The pio was associated with the order (association formed). - order was associated with Supplier (association formed); (To add it to the historical order records of the current supplier). - order was associated with the Branch (association formed); (To add it to the historical order records of the current branch).


:orderController
periodicOrderDAO
:supplierDAO
:orderDAO
:IMOrders
:SQLiteDataSource
executePriodicOrder(orderID))
getPeriodicOrderByID(orderID)
s = getSupplier(supplierID)
opt
[no s in IMSuppliers]
srec = sql(select supplier by sid)
s = supplier(srec)
s:Supplier
s = add(sid, s)
s
updateProductAmount(productsANDAmount)
loop
[for each product]
deleteAmount(productID, amount)
a:Agreement
getTotalPrice(productsANDAmount)
getDeliveryDate()
createOrder(branchID, supplierID, productsANDAmount, totalPrice, orderDate, deliveryDate))
o = Order(...)
o:order
add(o)

ERD:

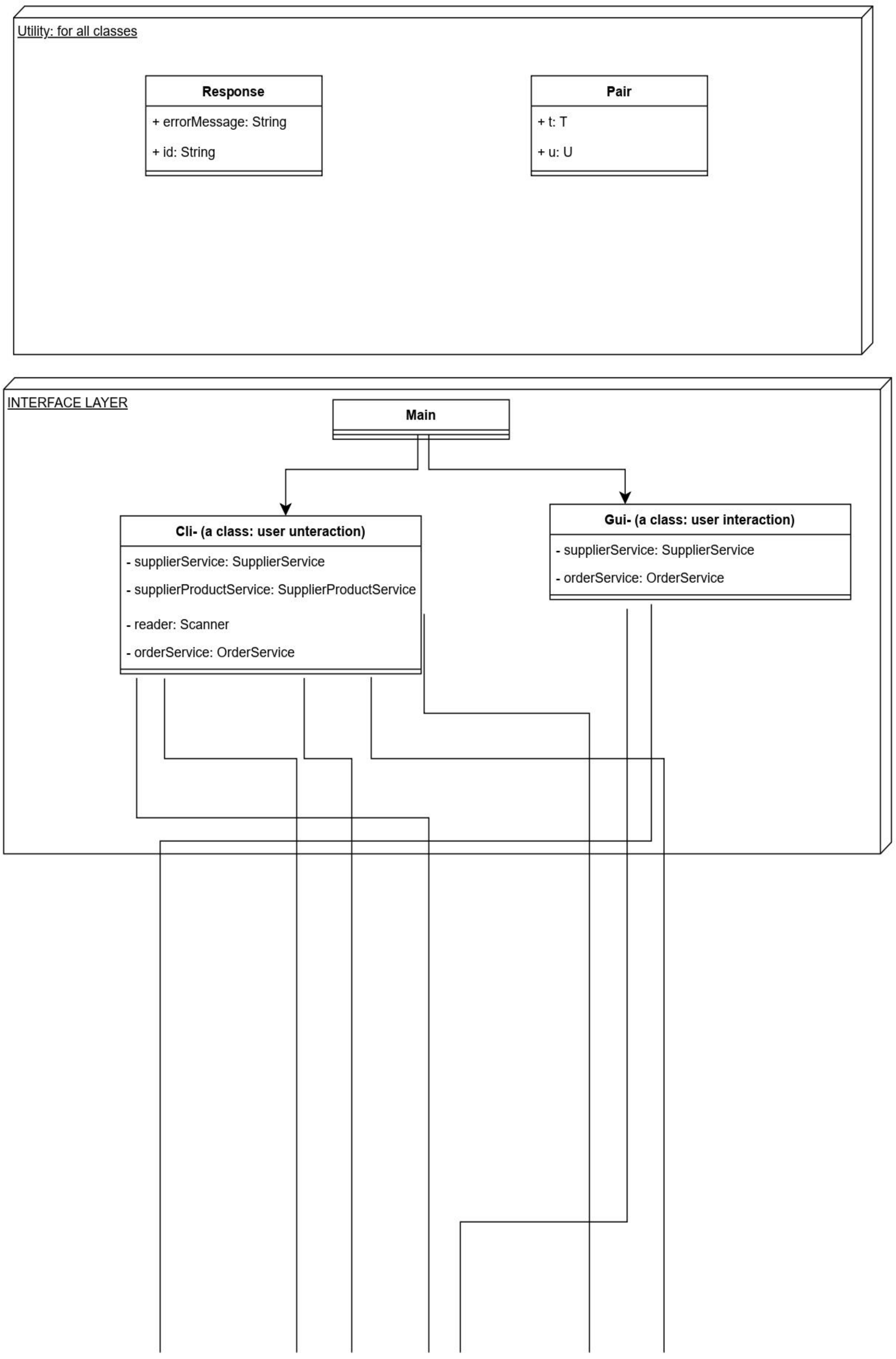


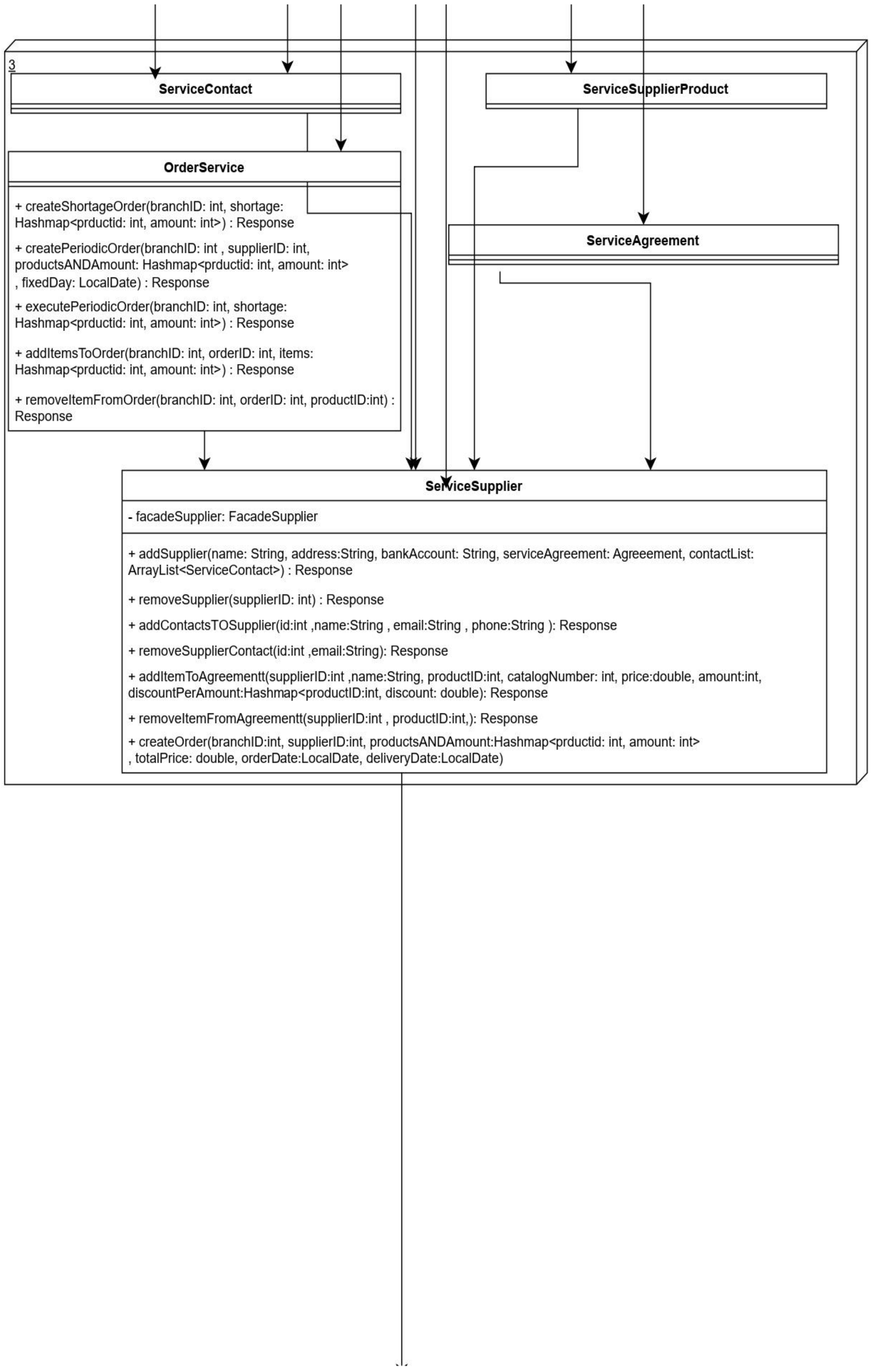
productId
amount
itemsInOrder
n
has
1
supplierId
peiodicOrderId
dayToCreate
periodicOrder
branch
n
address
supplierId
name
bankAccount
supplier
1
has
supplyMethod
1
has
1
n
has
n
1
contact
name
email
phoneNumber
agreement
paymentType
selfSupply
id
deliveryDays
day
1
has
n
has
supplierId
has
1
price
supplierProduct
amount
1
catalogNumber
name
productId
discountPerAmount
discount
amount
n
productId
amount
itemsInOrder
n
has
1
order
branch
creationDate
supplierId
orderId
Transfer To
Categories
categoryID
categoryName
3
has
has
productID
0..n
0..n
Products
productName
manufacturer
wight



has
branchID
branchName
Branches
discountID
branchID
BranchDiscounts
productID
amount
endDate
startDate
categoryID
has
branchID
itemID
expiredDate
supplierID
priceFromSupplier
Items
productID
PriceInBranch
PriceAfterDiscount
Damaged
status
Storage
defectiveDiscription
Expired
sold
Store
arrivalDate
has
ProdcutMinAmountInBranch
minAmount
productID
orderStatus
branchID
has
reportID
branchID
itemID
expiredDate
defectiveDiscription
productID
supplierID
Defective
priceFromSupplier
status
Expired
Damaged
reportID
BranchReports
reportType
reportDate
amount
reportID
Missing
Weekly
branchID
branchID
reportID
productID
categoryID
amountToOrder
amount
productID

Class Diagram:


Utility: for all classes
Response
+ errorMessage: String
+ id: String
Pair
+ t: T
+ u: U
INTERFACE LAYER
Main
Cli- (a class: user unteraction)
- supplierService: SupplierService
- supplierProductService: SupplierProductService
- reader: Scanner
- orderService: OrderService
Gui- (a class: user interaction)
- supplierService: SupplierService
- orderService: OrderService



ServiceContact

ServiceSupplierProduct

OrderService

+ createShortageOrder(branchID: int, shortage: Hashmap<prductid: int, amount: int>) : Response

+ createPeriodicOrder(branchID: int , supplierID: int, productsANDAmount: Hashmap<prductid: int, amount: int> , fixedDay: LocalDate) : Response

+ executePeriodicOrder(branchID: int, shortage: Hashmap<prductid: int, amount: int>) : Response

+ addItemsToOrder(branchID: int, orderID: int, items: Hashmap<prductid: int, amount: int>) : Response

+ removeItemFromOrder(branchID: int, orderID: int, productID:int) : Response

ServiceAgreement

ServiceSupplier

- facadeSupplier: FacadeSupplier

+ addSupplier(name: String, address:String, bankAccount: String, serviceAgreement: Agreeement, contactList: ArrayList<ServiceContact>) : Response

+ removeSupplier(supplierID: int) : Response

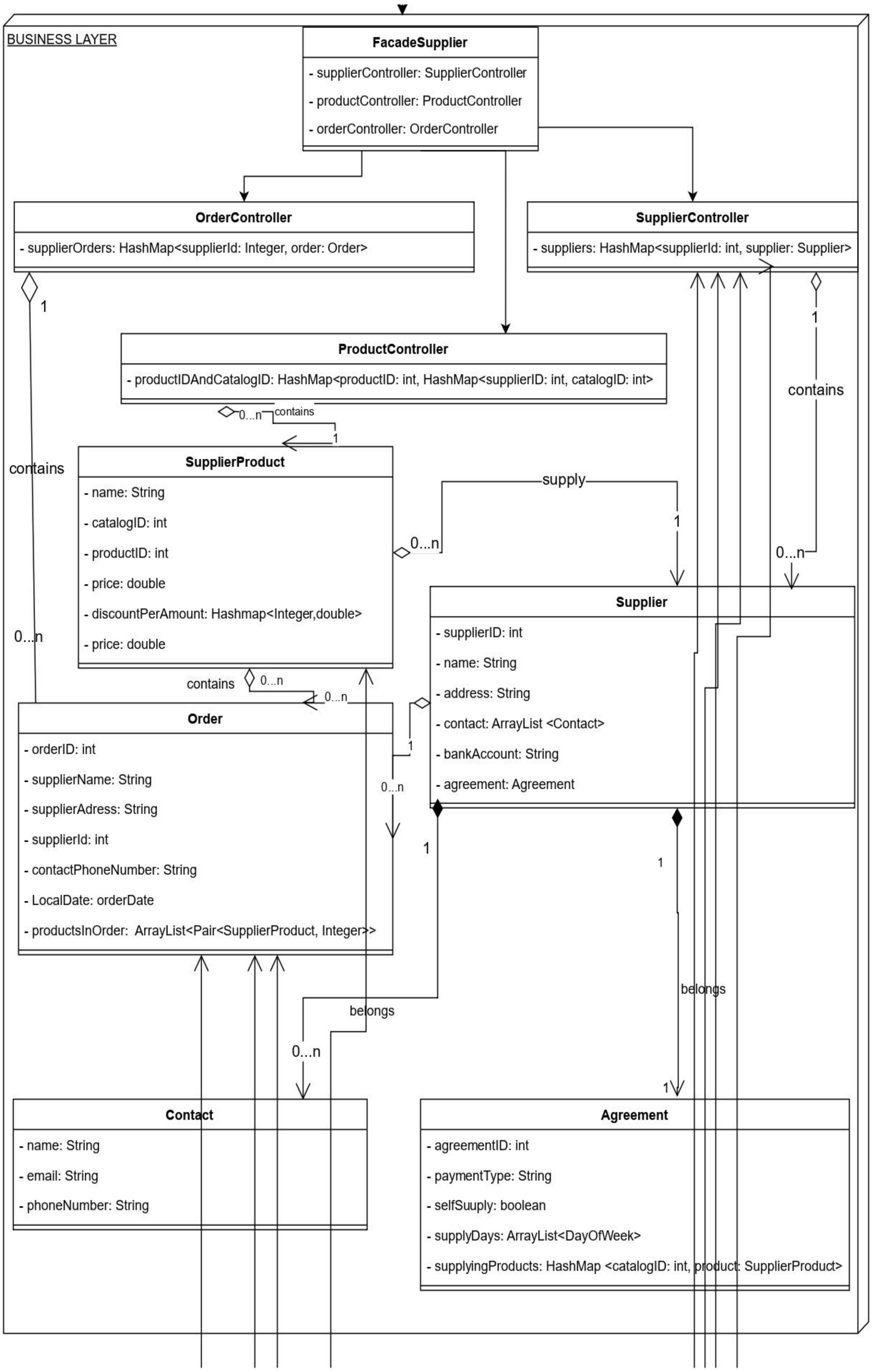
+ addContactsTOSupplier(id:int ,name:String , email:String , phone:String): Response

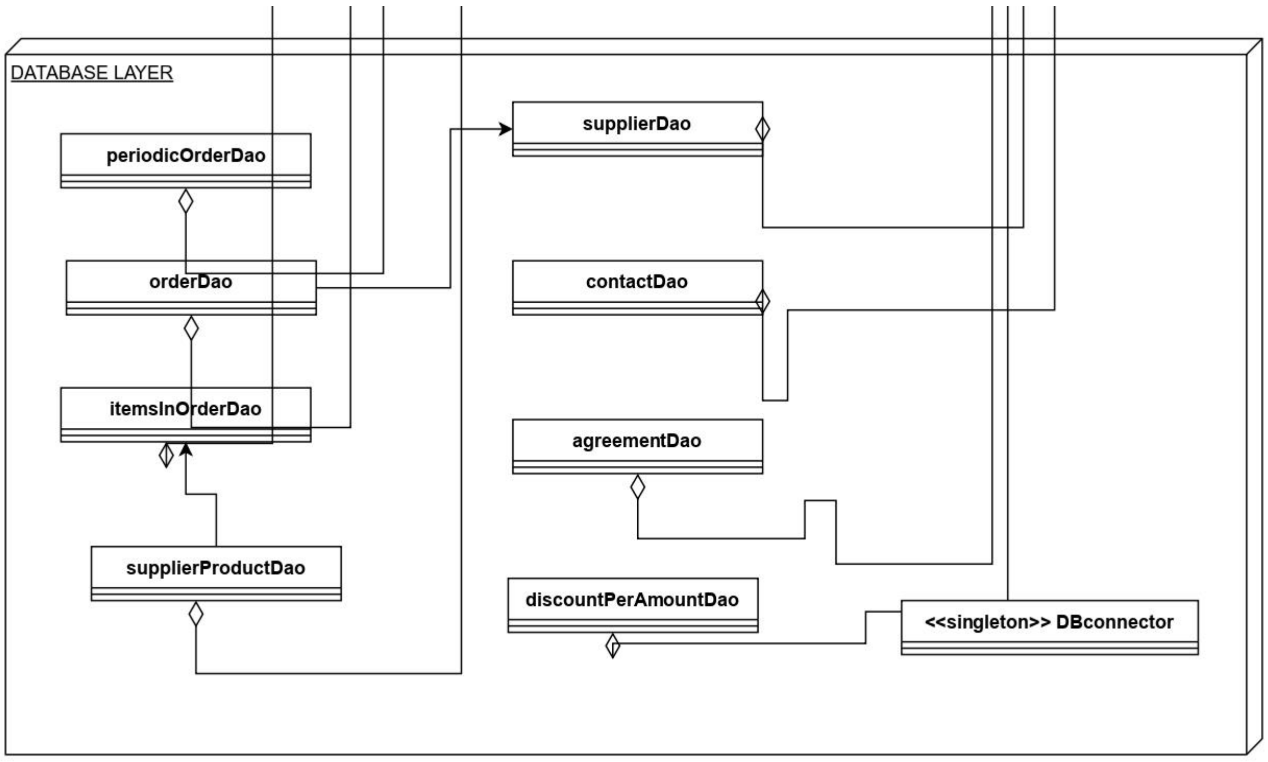
+ removeSupplierContact(id:int ,email:String): Response

+ addItemToAgreementt(supplierID:int ,name:String, productID:int, catalogNumber: int, price:double, amount:int, discountPerAmount:Hashmap<productID:int, discount: double): Response

+ removeItemFromAgreementt(supplierID:int , productID:int,): Response

+ createOrder(branchID:int, supplierID:int, productsANDAmount:Hashmap<prductid: int, amount: int> , totalPrice: double, orderDate:LocalDate, deliveryDate:LocalDate)

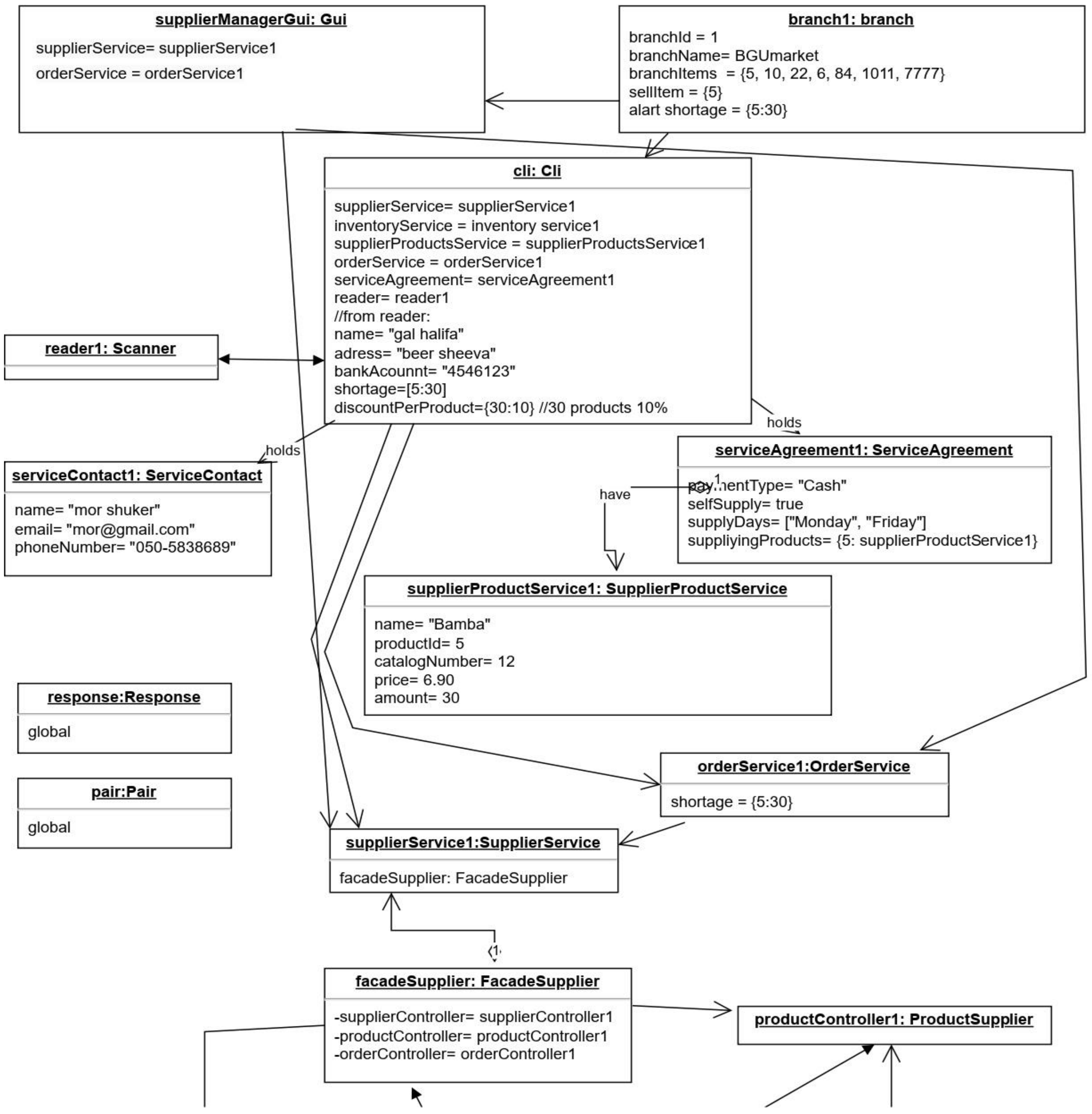

BUSINESS LAYER
FacadeSupplier
- supplierController: SupplierController
- productController: ProductController
- orderController: OrderController
OrderController
- supplierOrders: HashMap<supplierId: Integer, order: Order>
SupplierController
- suppliers: HashMap<supplierId: int, supplier: Supplier>
ProductController
- productIDAndCatalogID: HashMap<productID: int, HashMap<supplierID: int, catalogID: int>
SupplierProduct
- name: String
- catalogID: int
- productID: int
- price: double
- discountPerAmount: Hashmap<Integer,double>
- price: double
Supplier
- supplierID: int
- name: String
- address: String
- contact: ArrayList <Contact>
- bankAccount: String
- agreement: Agreement
Order
- orderID: int
- supplierName: String
- supplierAdress: String
- supplierId: int
- contactPhoneNumber: String
- LocalDate: orderDate
- productsInOrder: ArrayList<Pair<SupplierProduct, Integer>>
Contact
- name: String
- email: String
- phoneNumber: String
Agreement
- agreementID: int
- paymentType: String
- selfSuuply: boolean
- supplyDays: ArrayList<DayOfWeek>
- supplyingProducts: HashMap <catalogID: int, product: SupplierProduct>
contains
1
0...n
contains
1
0...n
contains
0...n
1
supply
1
0...n
contains
0...n
0...n
1
0...n
1
belongs
0...n
1
belongs
1

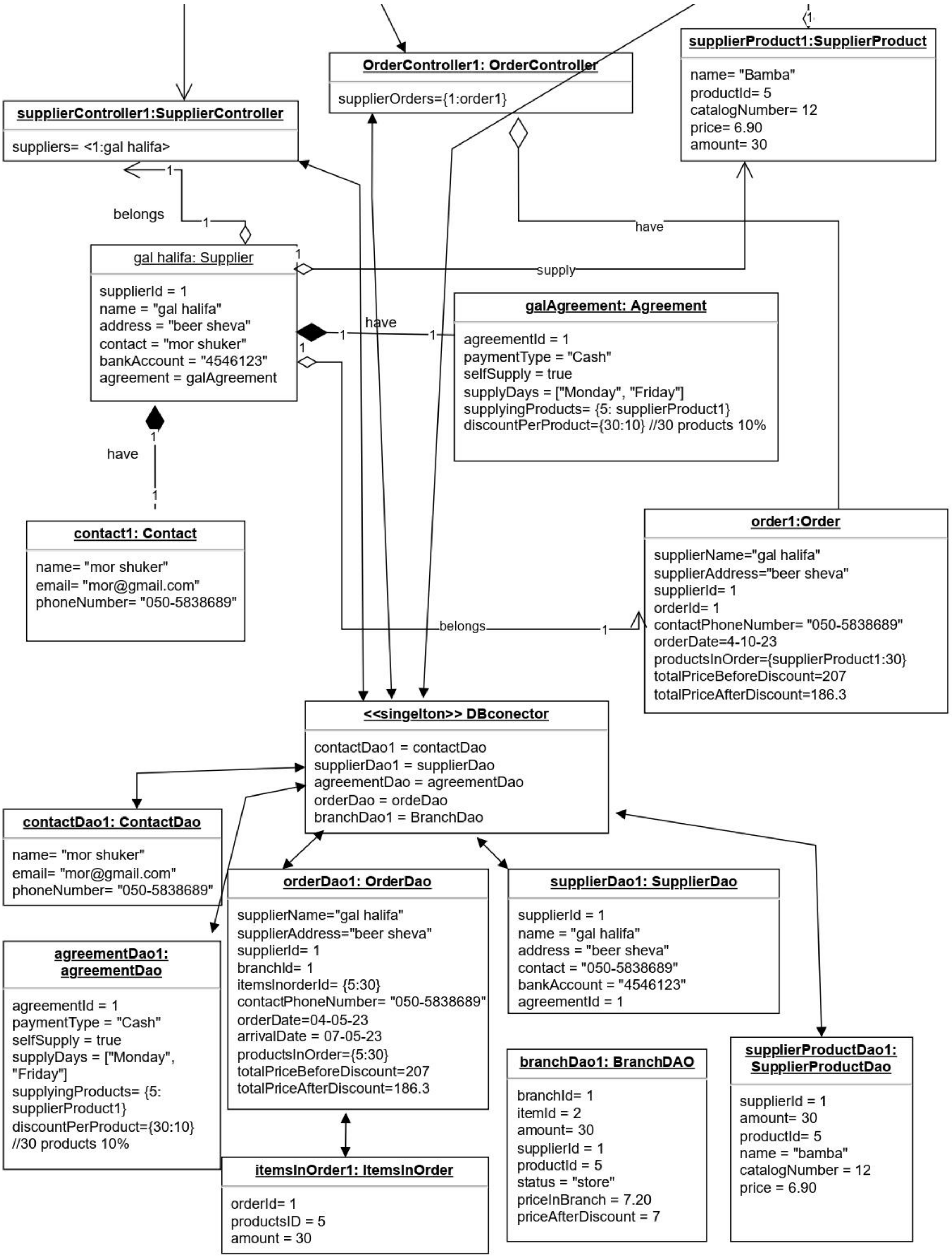

DATABASE LAYER
periodicOrderDao
orderDao
itemsInOrderDao
supplierProductDao
supplierDao
contactDao
agreementDao
discountPerAmountDao
<<singleton>> DBconnector

Object Diagram

תרחיש ראשון:
בתאריך 4.5.23 כחלק מבדיקות האינטגרציה עם מחלקת מלאי החליטו צוות הסופר לבדוק האם מערכת- "הזמן בשעת חוסר" עובדת כראוי.
מנהל הסניף ביצע קנייה של 30 מוצרים בעלי מספר קטגורי: 5 בכדי לבחון האם ההזמנה תיווצר באופן אוטומטי.
כעת, עבור סניף BGU הזמנה מסוג "חוסר" אמורה להתבצע. ומפני שגל זהו הספק היחידי במערכת ההזמנה נוצרה עבור ספק גל- ID ייחודי: 1

כמתואר בתרשים גל מספק במבות לסופר- לי, ברשותו 30 במבות.
כחלק מההתקשרות עם הסופר לגל יש הסכם עם הסופר (פרטים מתוארים בתרשים ובדרישות המערכת).
המערכת קיבלה מצוות המחסן ומסניף BGU התראה דחופה למילוי חוסרים בכדי שהסופר ימשיך לתת שירות לכלל לקוחותיו.
צוות המחסן העביר בקשה לגל בכדי לרכוש את המוצרים ממנו דרך המערכת האוטומטית של הסופר.
ההזמנה מהספק עובדה במערכת ונוצרה הזמנה חדשה מגל בנוסף ההזמנה נשמרה במסד נתונים וזאת על מנת שבעלי הסופר תהיה גישה לכל ההזמנות שבוצעו בסניף- כעת ישנה רק הזמנת חוסרים אחת.
גל העניק 10 אחוז הנחה עבור 30 הבמבות שנרכשו ממנו כפי מתואר בהסכם- בנוסף גל עדכן את כמות הפריטים הזמינים לרכישה ברגע שהתשלום עבר ולפיכך הסופר התעדכן במלאי הזמין החדש שמתעדכן בהתאם לרכישות השונות.
בנוסף, לאחר שהצוות אישר את ההזמנה, ההזמנה התעדכנה במסד הנתונים של הסופר ששומר את מוצרי הסופר הזמינים במערכת.


supplierManagerGui: Gui
supplierService= supplierService1
orderService = orderService1
branch1: branch
branchId = 1
branchName= BGUmarket
branchItems = {5, 10, 22, 6, 84, 1011, 7777}
sellItem = {5}
alart shortage = {5:30}
cli: Cli
supplierService= supplierService1
inventoryService = inventory service1
supplierProductsService = supplierProductsService1
orderService = orderService1
serviceAgreement= serviceAgreement1
reader= reader1
//from reader:
name= "gal halifa"
adress= "beer sheeva"
bankAcounnt= "4546123"
shortage=[5:30]
discountPerProduct={30:10} //30 products 10%
reader1: Scanner
holds
serviceContact1: ServiceContact
name= "mor shuker"
email= "mor@gmail.com"
phoneNumber= "050-5838689"
holds
serviceAgreement1: ServiceAgreement
paymentType= "Cash"
selfSupply= true
supplyDays= ["Monday", "Friday"]
suppliyingProducts= {5: supplierProductService1}
have
1
supplierProductService1: SupplierProductService
name= "Bamba"
productId= 5
catalogNumber= 12
price= 6.90
amount= 30
response:Response
global
pair:Pair
global
orderService1:OrderService
shortage = {5:30}
supplierService1:SupplierService
facadeSupplier: FacadeSupplier
1
facadeSupplier: FacadeSupplier
-supplierController= supplierController1
-productController= productController1
-orderController= orderController1
productController1: ProductSupplier


supplierController1:SupplierController
suppliers= <1:gal halifa>
OrderController1: OrderController
supplierOrders={1:order1}
supplierProduct1:SupplierProduct
name= "Bamba"
productId= 5
catalogNumber= 12
price= 6.90
amount= 30
1
belongs
1
1
have
gal halifa: Supplier
supplierId = 1
name = "gal halifa"
address = "beer sheva"
contact = "mor shuker"
bankAccount = "4546123"
agreement = galAgreement
1
supply
have
1
1
1
galAgreement: Agreement
agreementId = 1
paymentType = "Cash"
selfSupply = true
supplyDays = ["Monday", "Friday"]
supplyingProducts= {5: supplierProduct1}
discountPerProduct={30:10} //30 products 10%
1
have
1
contact1: Contact
name= "mor shuker"
email= "mor@gmail.com"
phoneNumber= "050-5838689"
order1:Order
supplierName="gal halifa"
supplierAddress="beer sheva"
supplierId= 1
orderId= 1
contactPhoneNumber= "050-5838689"
orderDate=4-10-23
productsInOrder={supplierProduct1:30}
totalPriceBeforeDiscount=207
totalPriceAfterDiscount=186.3
belongs
1
<<singelton>> DBconector
contactDao1 = contactDao
supplierDao1 = supplierDao
agreementDao = agreementDao
orderDao = ordeDao
branchDao1 = BranchDao
contactDao1: ContactDao
name= "mor shuker"
email= "mor@gmail.com"
phoneNumber= "050-5838689"
orderDao1: OrderDao
supplierName="gal halifa"
supplierAddress="beer sheva"
supplierId= 1
branchId= 1
itemsInorderId= {5:30}
contactPhoneNumber= "050-5838689"
orderDate=04-05-23
arrivalDate = 07-05-23
productsInOrder={5:30}
totalPriceBeforeDiscount=207
totalPriceAfterDiscount=186.3
supplierDao1: SupplierDao
supplierId = 1
name = "gal halifa"
address = "beer sheva"
contact = "050-5838689"
bankAccount = "4546123"
agreementId = 1
agreementDao1: agreementDao
agreementId = 1
paymentType = "Cash"
selfSupply = true
supplyDays = ["Monday", "Friday"]
supplyingProducts= {5: supplierProduct1}
discountPerProduct={30:10} //30 products 10%
branchDao1: BranchDAO
branchId= 1
itemId = 2
amount= 30
supplierId = 1
productId = 5
status = "store"
priceInBranch = 7.20
priceAfterDiscount = 7
supplierProductDao1: SupplierProductDao
supplierId = 1
amount= 30
productId= 5
name = "bamba"
catalogNumber = 12
price = 6.90
itemsInOrder1: ItemsInOrder
orderId= 1
productsID = 5
amount = 30

תרחיש שני:

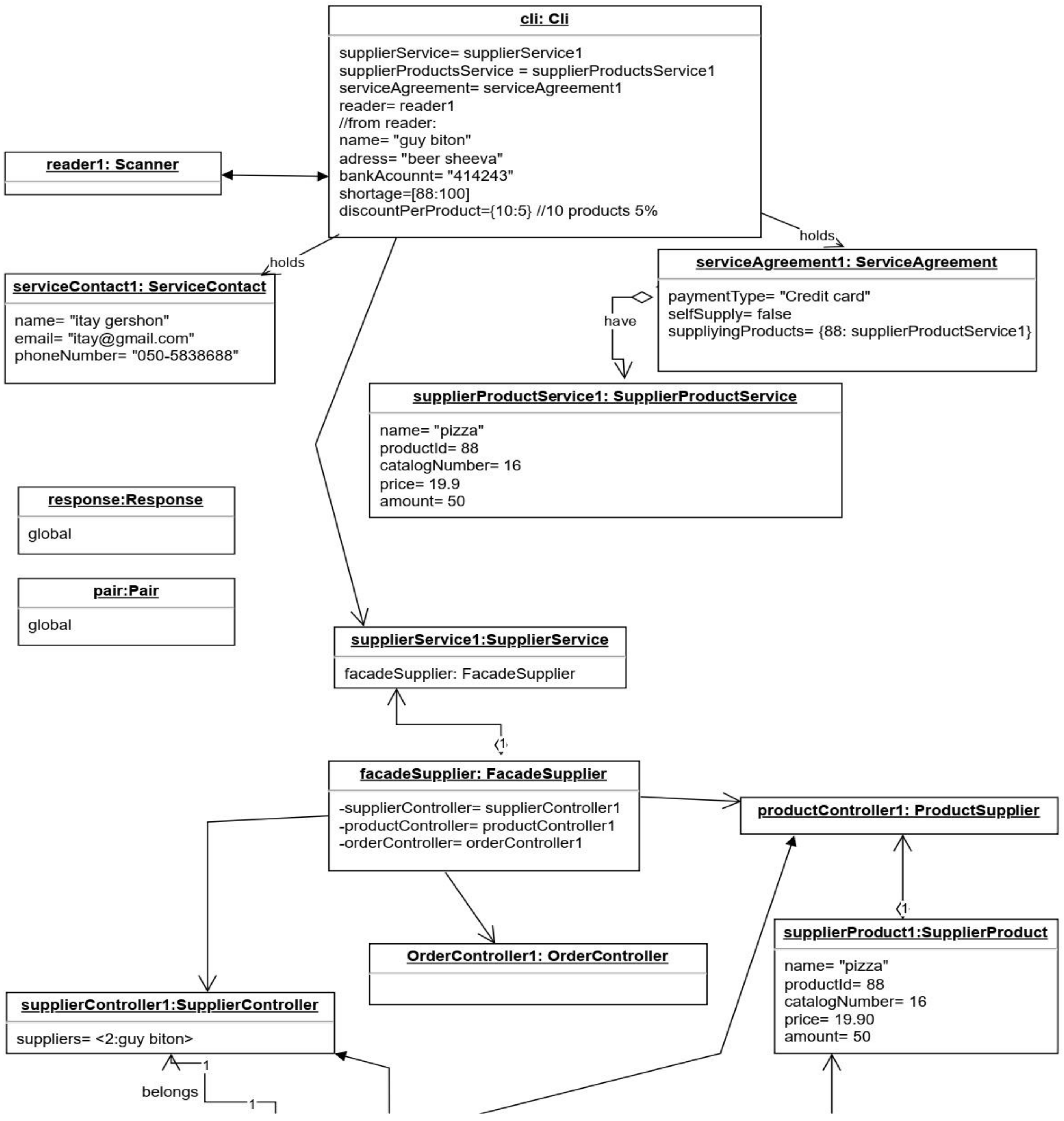
לאחר שההזמנה עקב חוסר בוצעה בהצלחה ונקלטה במערכת, הצוות הסיר את הספק בעל מס' ספק: 1, ומיד לאחר מכן הוסיפו ספק חדש למערכת בשם- "גיא כהן". גיא ידוע כספק גדול של פיצות בישראל ולכן החליט לעבור עם הסופר בכדי להרחיב את שירותיו גם בסופר המקומי BGU.

לאחר ההסרה יש לשים לב שמספר הספק ומספר ההסכם עלו ל- 2.

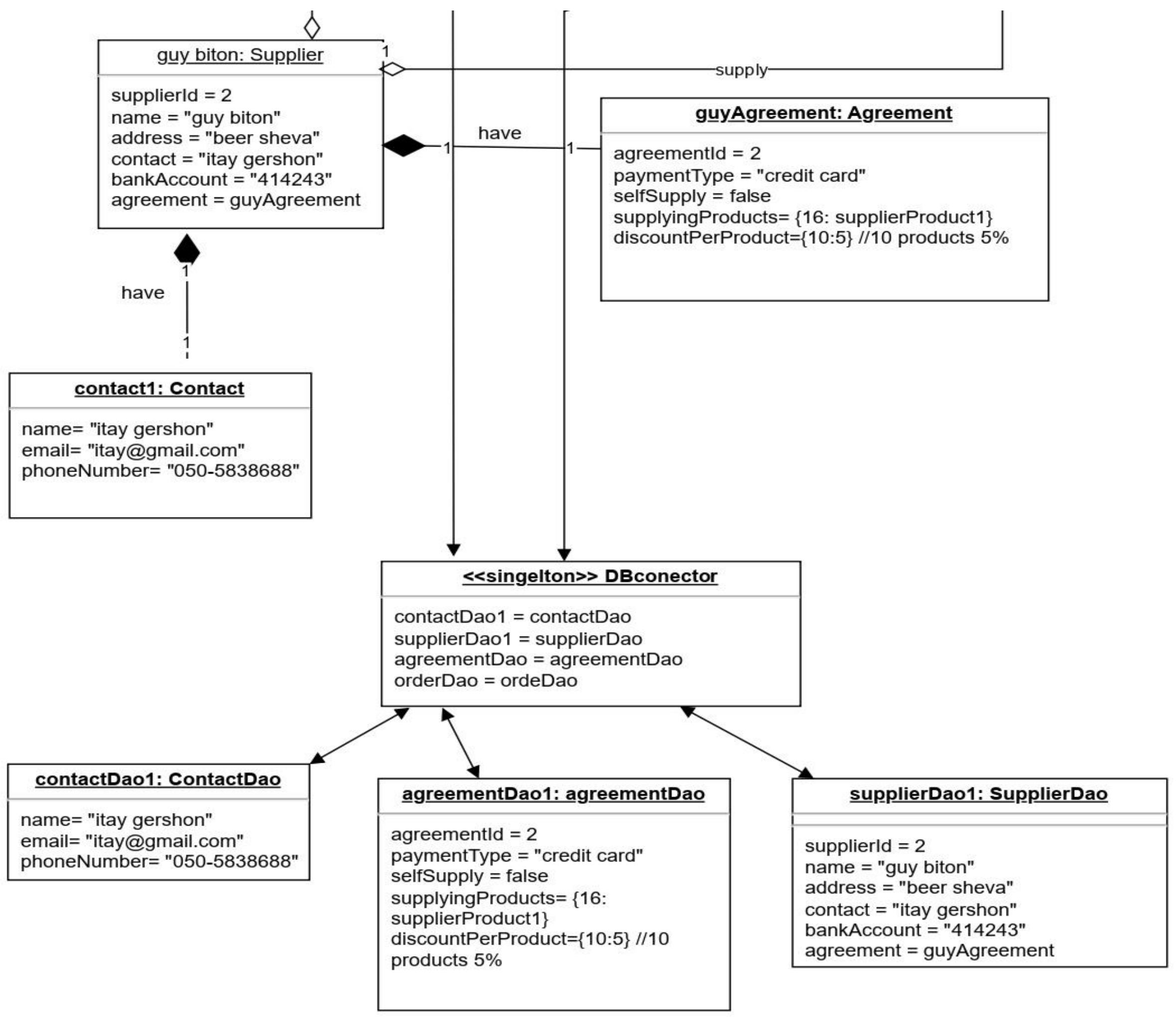
כעת הגיעו צוות הסופר בכדי לבצע הזמנת חוסרים נוספת- 100 במבות.

הצוות טען את הנתונים אודות הספק- גיא, בתרחיש זה אנו מסתכלים על המערכת בשלב העלייה שלה כאשר הנתונים רק נטענו באמצעות מסד הנתונים של החברה.

כצפוי ההזמנה צפויה לא להתבצע מפני שגיא אינו מספק במבות בכלל אלא רק פיצה. לכן המערכת הציגה הודעה שהספק אינו מחזיק במוצר המבוקש וההזמנה לא בוצעה- כפי שניתן לראות בתרשים



cli: Cli
supplierService= supplierService1
supplierProductsService = supplierProductsService1
serviceAgreement= serviceAgreement1
reader= reader1
//from reader:
name= "guy biton"
adress= "beer sheeva"
bankAcounnt= "414243"
shortage=[88:100]
discountPerProduct={10:5} //10 products 5%
reader1: Scanner
holds
serviceContact1: ServiceContact
name= "itay gershon"
email= "itay@gmail.com"
phoneNumber= "050-5838688"
holds
serviceAgreement1: ServiceAgreement
paymentType= "Credit card"
selfSupply= false
suppliyingProducts= {88: supplierProductService1}
have
supplierProductService1: SupplierProductService
name= "pizza"
productId= 88
catalogNumber= 16
price= 19.9
amount= 50
response:Response
global
pair:Pair
global
supplierService1:SupplierService
facadeSupplier: FacadeSupplier
1
facadeSupplier: FacadeSupplier
-supplierController= supplierController1
-productController= productController1
-orderController= orderController1
productController1: ProductSupplier
1
supplierProduct1:SupplierProduct
name= "pizza"
productId= 88
catalogNumber= 16
price= 19.90
amount= 50
OrderController1: OrderController
supplierController1:SupplierController
suppliers= <2:guy biton>
1
belongs
1



guy biton: Supplier
supplierId = 2
name = "guy biton"
address = "beer sheva"
contact = "itay gershon"
bankAccount = "414243"
agreement = guyAgreement
1
supply
have
1
1
guyAgreement: Agreement
agreementId = 2
paymentType = "credit card"
selfSupply = false
supplyingProducts= {16: supplierProduct1}
discountPerProduct={10:5} //10 products 5%
1
have
1
contact1: Contact
name= "itay gershon"
email= "itay@gmail.com"
phoneNumber= "050-5838688"
<<singelton>> DBconector
contactDao1 = contactDao
supplierDao1 = supplierDao
agreementDao = agreementDao
orderDao = ordeDao
contactDao1: ContactDao
name= "itay gershon"
email= "itay@gmail.com"
phoneNumber= "050-5838688"
agreementDao1: agreementDao
agreementId = 2
paymentType = "credit card"
selfSupply = false
supplyingProducts= {16: supplierProduct1}
discountPerProduct={10:5} //10 products 5%
supplierDao1: SupplierDao
supplierId = 2
name = "guy biton"
address = "beer sheva"
contact = "itay gershon"
bankAccount = "414243"
agreement = guyAgreement

Requirements:

Table 1: Requirements

ID	Module	Functional/ not functional	Description	Priority	Risk	Status
1	Suppliers	Functional	The system must have the ability to create supplier accounts.	MH	LR	Done
2	Suppliers	Functional	The system must save the essential details about the suppliers- bank account, company id, name, last name, financial guarantee of the supplier, terms of payment, contact, supplier agreement	MH	LR	Done
3	Suppliers	Functional	the system will save for every supplier his unique catalog id for products in the super	MH	low	Done
4	Suppliers	Functional	The system needs to manage supplier information such as the items they can supply, along with their prices and catalog numbers	MH	HR	Done
5	Suppliers	Functional	for every supplier the system will save an agreement contact	MH	HR	Done
6	Suppliers	Functional	The system may store discounts for suppliers who offer discounts based on the number of products in an order	NTH	HR	Done
7	Suppliers	Functional	The system will store discounts for suppliers based on the total price of an order	NTH	HR	Done
8	Suppliers	Functional	The system will store discounts for suppliers based on negotiated agreements where different discounts are offered for different order quantities	NTH	LR	Done
9	Suppliers	Functional	The system must have the capability to create new orders from suppliers	MH	LR	Done

10	Suppliers	Functional	The system needs to save for each order who is the deliver, "Super Lee" or the supplier	MH	LR	Done
11	Suppliers	Functional	The system should save all the past orders from the suppliers	MH	HR	Done
12	Suppliers	Functional	The system should save all the past orders from the suppliers	MH	LR	Done
13	Suppliers and Inventory	Functional	the system will allows the user to edit supplier's details	MH	LR	Done
14	Suppliers and Inventory	Functional	The system will support periodic ordering, where on specific days it will create an order based on the inventory demand	MH	HR	Done
15	Suppliers and Inventory	Functional	The system will support shortage orders, whereby based on inventory leaks, the system will automatically create an order to refill the inventory	MH	HR	Done
16	Suppliers and Inventory	Functional	The system will support editing an order-periodic order at least a day before it is packaged	MH	HR	Done
17	Suppliers and Inventory	Non-functional	The system should support graphic user interface visualization- GUI	MH	HR	Done
18	Suppliers and Inventory	Functional	The System should support 3 different menus for 3 different roles: Storekeeper, Supplier Manager and Store Manager	NTH	LR	DONE
19	Suppliers and Inventory	Functional	The system should support control of inventory and orders in the storekeeper menu	MH	LR	DONE
20	Suppliers and Inventory	Functional	The system should support control of supplier management in the supplier manager menu	MH	LR	DONE
21	Suppliers and Inventory	Functional	The system should support access to data and reports in the store manager menu	MH	LR	DONE

22	Suppliers and Inventory	Non-functional	The system should support one data base for both modules	NTH	LR	**DONE**

דרישות 17-22 נוספו

ID	Module	Functional / Non - Functional	Description	Priority	Risk	Status
1	Inventory	Functional	The system shall allow entry of deficiencies by the store employees	MH	Low	**Done**
2	Inventory	Functional	The system shall allow the issuing of a detailed report that needs to be ordered based on the existing stock	MH	High	**Done**
3	Inventory	Functional	The system shall give an advance warning based on a drop below a certain minimum amount	MH	Low	**Done**
4	Inventory	Functional	The system shall save for each item in stock the location, manufacturer, current quantity, quantity on the store shelves and quantity in the warehouse	MH	High	**Done**
5	Inventory	Functional	The system can update on Mondays and Thursdays about stock counts	NTH	Low	**Done**
6	Inventory	Functional	The system shall track and record the cost prices (from the supplier) of each item and the price at which it is sold in the store	MH	High	**Done**
7	Inventory	Functional	The system support provision of promotions and a temporary change of product prices according to certain dates	MH	High	**Done**
8	Inventory	Functional	The system shall save the item data according to categories and subcategories	MH	High	**Done**

9	Inventory	Functional	The system shall generate an inventory report at least once a week according to a category or several categories that will be detailed in the report	MH	Low	**Done**
10	Inventory	Functional	The system shall allow the creation of an inventory report by categories	MH	High	**Done**
11	Inventory	Functional	The system shall monitor the presence of damaged or expired items in accordance with reports from store employees and produce a damaged items report	MH	Low	**Done**
12	Inventory	Functional	The system shall allow the addition of damaged or expired items to the report	MH	High	**Done**
13	Inventory	Functional	The system can store information about the validity of the products if they exist	MH	Low	**Done**
14	Inventory	Functional	The system must save all the reports that are produced	NTH	Low	**Done**
15	Inventory	Functional	The system can receive an item code and tell the user where it is in the warehouse or store	NTH	Low	**Done**
16	Inventory	Functional	The system shall allow adding a new product or category	MH	Low	**Done**

Table 2: Close Questions - self assumptions

Number	Question	Answer
1	How the system will pay on order to supplier?	cash, transaction, net dom 30, net dom 60
2	If the client nee d to login in the model?	No, suppliers aren't workers of the super
3	Where the system will be in use?	In the Storage offices
4	how many suppliers can be registered in the system?	As many as the shift workers want
5	how the discount will calculate?	first, discount per product s in percentage, after that discount per total orders price and lastly discount for amount of products
6	will the super be open on Saturday?	On Saturday the supper will be close.
7	is there a difference between suppliers, Like type of them?	In the system there is no difference between the suppliers.
8	will the system support different kind of currencies?	At that point the system works only in ISRAEL base on that the currency the system support is only Israeli new shekel.
9	Does the type of employee who will count the inventory important?	The type of employee is not important.
10	For each product what is the minimal quantity which must a notification be given for a new order?	Quantity given by the user.
11	Is a notification made in real time after a product is sold or after a stock count?	The notification is made in real time after the sale of a product.
12	Is a notification made in real time after a product is sold or after a stock count?	The information on each item can change.
13	What is the need according to which we would like to produce an inventory report?	Upon request from the user.
14	Is the category for the report given or chosen randomly or chosen according to a certain order?	For each item that is entered into the system, there will be requirement to enter the category to which it belongs.
15	How are the discount percentages applied at the supermarket? For example: Is a discount plus a discount on a category a double sale?	The highest discount is the chosen one (considering its validity according to the dates).
16	How often should the system produce a periodic report for damaged or expired products?	Whenever the user requests.

17	Is the price for the product the same in each branch?	The price is the same in every branch unless there are specific discounts in the branch for that product
18	Is the purchase price from a supplier the same for each product? (without discounts)	The price from a supplier can vary depending on the quantity you purchase or perform from the supplier.

<u>Concepts in the supplier – storage module:</u>

Supplier: Someone who supplies products to the store.
Agreement: An agreement between store to specific supplier that contains the delivery method, list of supplier products and their prices that the supplier supply to the store, also contains discounts he offer to the store.
Order: a list of products and there amounts that the supplier will provide, contains the supplier ID, the branch ID that the order belongs two, the order date and the estimated delivery date.
Periodic Order: An order that will repeatedly create on a specific day from specific supplier.
Supplier Product: A product that the supplier supply to the store.
Product: a product that the store will sell and manage.
Item: an instance of a product that will have its own expiration date and location in the store.

<u>Changes that happened in the supplier – storage module:</u>

1. Added data base, so both modules implemented DAL layer with DAO objects.
2. in the storage module we added functions that place order from supplier if there is shortage or make periodic order that will go repeatedly.
3. In the supplier module we added unique product id to merge product of suppliers with product of storage module.

<u>Assumptions for the supplier – storage module:</u>

- The system should not work on Saturday.
- The payment between supplier and super li will be made outside of the system.
- The system will be used in the storage office.
- The system will calculate a discount in the following way: First, calculate on the amount of product and then add the discount on the numbers of items in the order.
- The system will allow to make a report whenever a storekeeper would like.
- The system will account the damaged items and will not include them in the product amount.
- An item location is determined by if it is in the STOREAGE or in the STORE and by its shelf number.
- Every product can be in only one category.
- For every product the discount that applies to it, is the last discount that has entered the system.
- There are only category, subcategory, and sub subcategory.