SOFTWARE ENGINEERING (COM1028) - Summative coursework
Requirement set H
Student ID: 6595203

For the coursework, students were given one of the 8 requirements sets, each set consisting of 3 different requirements. The writer of this report has received set H for their coursework.

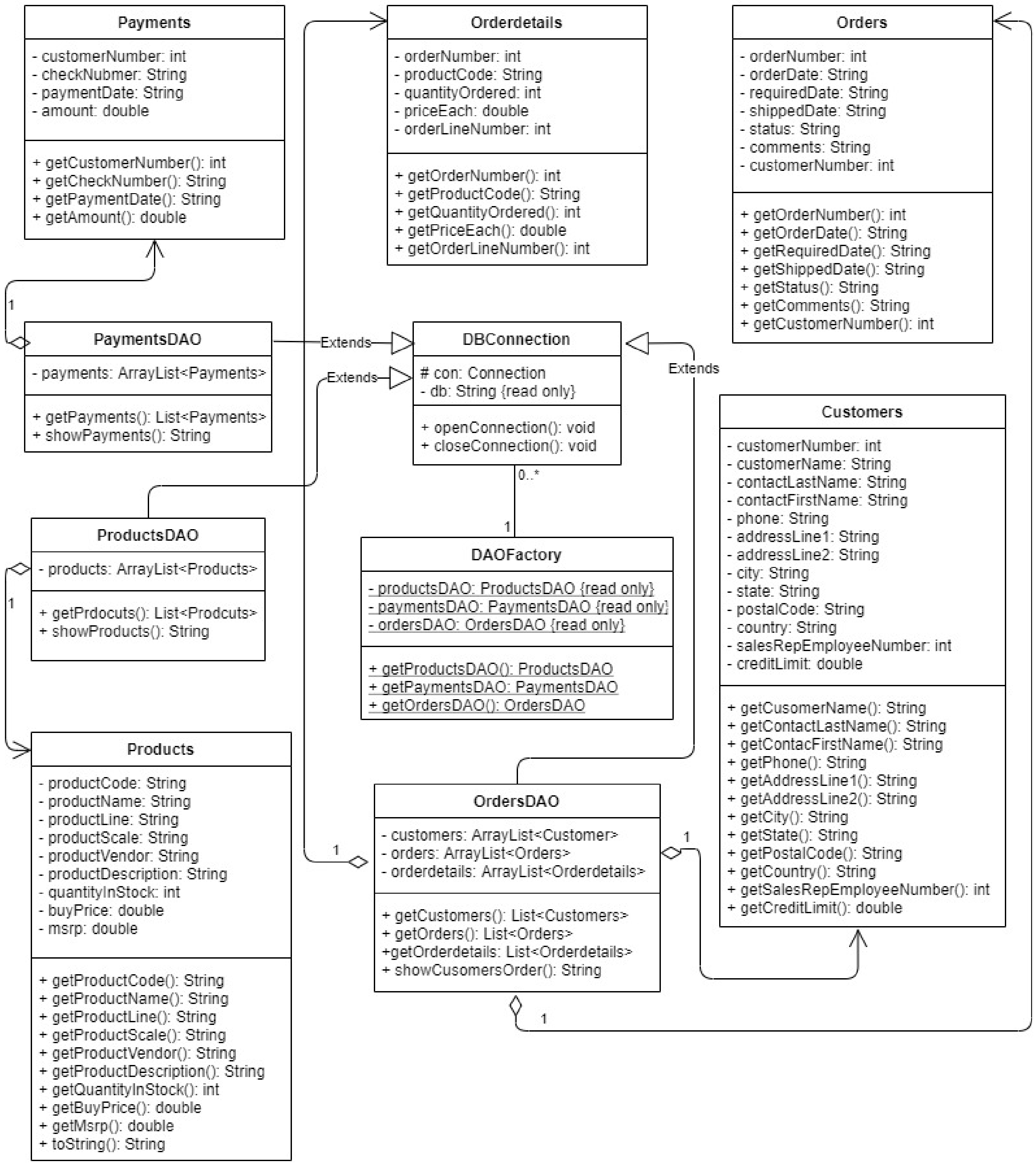
Payments
- customerNumber: int
- checkNubmer: String
- paymentDate: String
- amount: double
+ getCustomerNumber(): int
+ getCheckNumber(): String
+ getPaymentDate(): String
+ getAmount(): double
Orderdetails
- orderNumber: int
- productCode: String
- quantityOrdered: int
- priceEach: double
- orderLineNumber: int
+ getOrderNumber(): int
+ getProductCode(): String
+ getQuantityOrdered(): int
+ getPriceEach(): double
+ getOrderLineNumber(): int
Orders
- orderNumber: int
- orderDate: String
- requiredDate: String
- shippedDate: String
- status: String
- comments: String
- customerNumber: int
+ getOrderNumber(): int
+ getOrderDate(): String
+ getRequiredDate(): String
+ getShippedDate(): String
+ getStatus(): String
+ getComments(): String
+ getCustomerNumber(): int
1
PaymentsDAO
- payments: ArrayList<Payments>
+ getPayments(): List<Payments>
+ showPayments(): String
Extends
Extends
Extends
DBConnection
# con: Connection
- db: String {read only}
+ openConnection(): void
+ closeConnection(): void
0..*
1
Customers
- customerNumber: int
- customerName: String
- contactLastName: String
- contactFirstName: String
- phone: String
- addressLine1: String
- addressLine2: String
- city: String
- state: String
- postalCode: String
- country: String
- salesRepEmployeeNumber: int
- creditLimit: double
+ getCusomerName(): String
+ getContactLastName(): String
+ getContacFirstName(): String
+ getPhone(): String
+ getAddressLine1(): String
+ getAddressLine2(): String
+ getCity(): String
+ getState(): String
+ getPostalCode(): String
+ getCountry(): String
+ getSalesRepEmployeeNumber(): int
+ getCreditLimit(): double
ProductsDAO
- products: ArrayList<Products>
+ getPrdocuts(): List<Prodcuts>
+ showProducts(): String
1
DAOFactory
- productsDAO: ProductsDAO {read only}
- paymentsDAO: PaymentsDAO {read only}
- ordersDAO: OrdersDAO {read only}
+ getProductsDAO(): ProductsDAO
+ getPaymentsDAO: PaymentsDAO
+ getOrdersDAO(): OrdersDAO
Products
- productCode: String
- productName: String
- productLine: String
- productScale: String
- productVendor: String
- productDescription: String
- quantityInStock: int
- buyPrice: double
- msrp: double
+ getProductCode(): String
+ getProductName(): String
+ getProductLine(): String
+ getProductScale(): String
+ getProductVendor(): String
+ getProductDescription(): String
+ getQuantityInStock(): int
+ getBuyPrice(): double
+ getMsrp(): double
+ toString(): String
OrdersDAO
- customers: ArrayList<Customer>
- orders: ArrayList<Orders>
- orderdetails: ArrayList<Orderdetails>
+ getCustomers(): List<Customers>
+ getOrders(): List<Orders>
+getOrderdetails: List<Orderdetails>
+ showCusomersOrder(): String
1
1
1

Figure 1. UML Diagram of set H

The UML diagram is composed total of 10 classes (Figure 1): 1 database connection class, 5 simple classes mainly consisting of the getters and the setters, 3 data access object classes (DAOs) that act as helper code, and 1 data access object factory class. “DBConnection” class is a database connection class. It sets up a connection to the database using a try method allowing the programmer to directly access to the database from eclipse. It also contains opening and closing methods of the connection letting the user open and close the connection as needed. Secondly, the 5 simple classes-Products, Payments, Customers, Orders, and Orderdetails-are created to include the getters and the setters for the convenience in the DAO classes. Next, The DAO classes (ProductsDAO, PaymentDAO, and OrdersDAO) extend the DBConnection class to allow simple classes have access to data. In DAOs, the classes not only access to the database but also manipulate the data as the user’s need showing the desired information. Each DAO classes has at least one simple class. Lastly, DAOFactory calling all the DAOs together, allowing the programmer to test each DAOs with convenience in the main class. It has an association relationship with DBConnection.

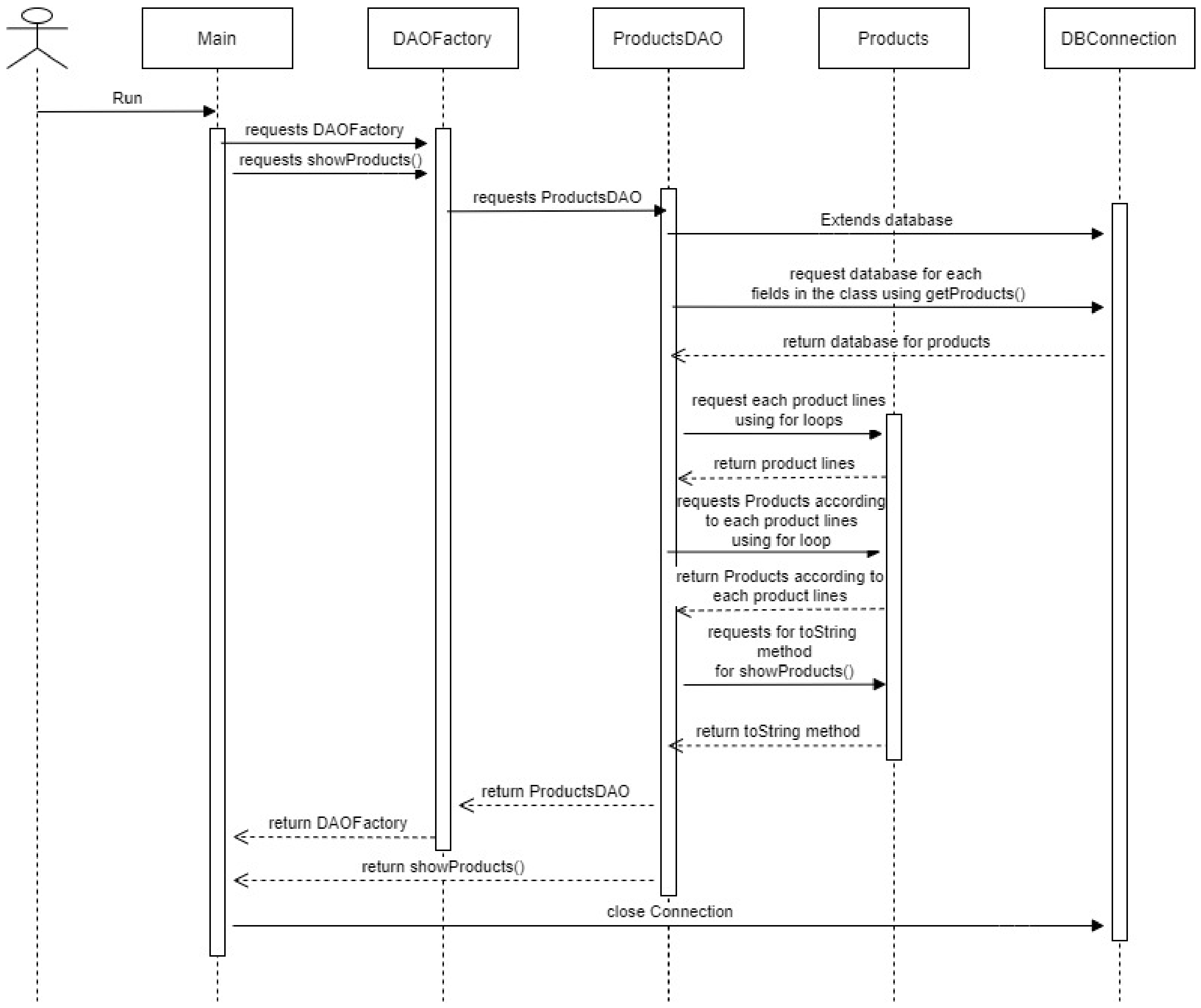
Main
DAOFactory
ProductsDAO
Products
DBConnection
Run
requests DAOFactory
requests showProducts()
requests ProductsDAO
Extends database
request database for each fields in the class using getProducts()
return database for products
request each product lines using for loops
return product lines
requests Products according to each product lines using for loop
return Products according to each product lines
requests for toString method for showProducts()
return toString method
return ProductsDAO
return DAOFactory
return showProducts()
close Connection

Figure 2. Sequence diagram of the first requirement of set H

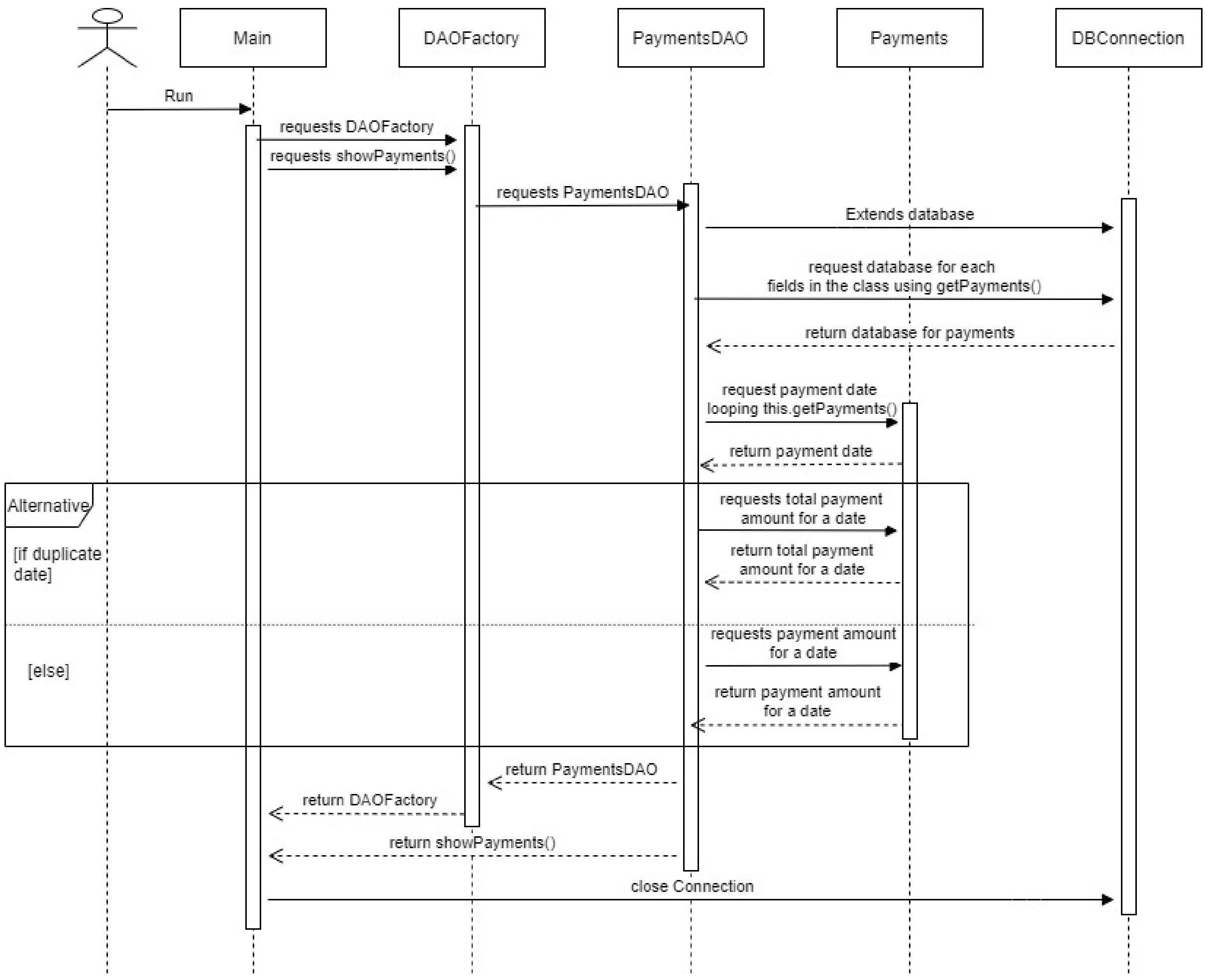
Main
DAOFactory
PaymentsDAO
Payments
DBConnection
Run
requests DAOFactory
requests showPayments()
requests PaymentsDAO
Extends database
request database for each fields in the class using getPayments()
return database for payments
request payment date looping this.getPayments()
return payment date
Alternative
[if duplicate date]
requests total payment amount for a date
return total payment amount for a date
[else]
requests payment amount for a date
return payment amount for a date
return PaymentsDAO
return DAOFactory
return showPayments()
close Connection

Figure 3. Sequence diagram of the second requirement of set H

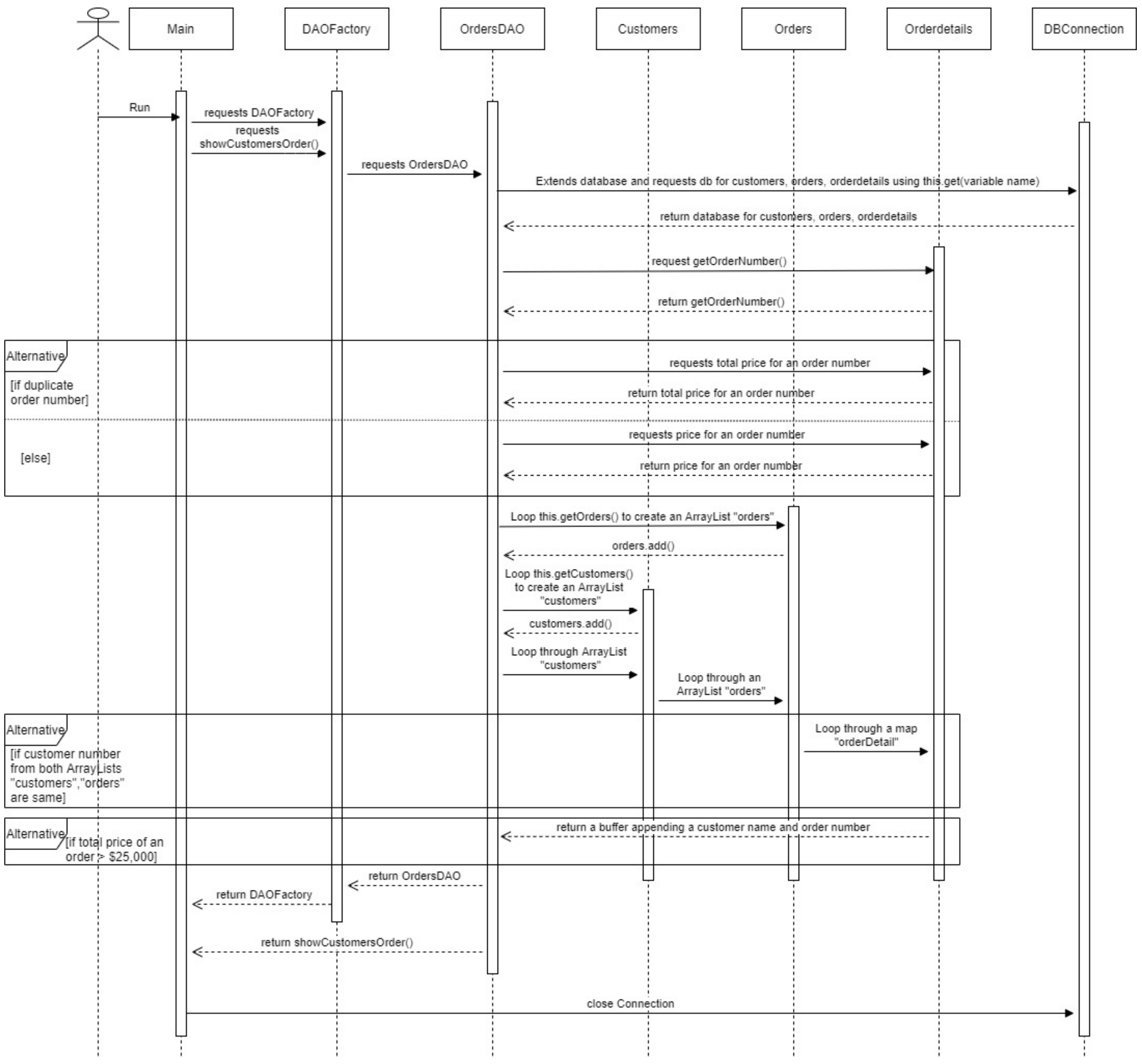
Main
DAOFactory
OrdersDAO
Customers
Orders
Orderdetails
DBConnection
Run
requests DAOFactory
requests showCustomersOrder()
requests OrdersDAO
Extends database and requests db for customers, orders, orderdetails using this.get(variable name)
return database for customers, orders, orderdetails
request getOrderNumber()
return getOrderNumber()
Alternative
[if duplicate order number]
requests total price for an order number
return total price for an order number
[else]
requests price for an order number
return price for an order number
Loop this.getOrders() to create an ArrayList "orders"
orders.add()
Loop this.getCustomers() to create an ArrayList "customers"
customers.add()
Loop through ArrayList "customers"
Loop through an ArrayList "orders"
Alternative
[if customer number from both ArrayLists "customers","orders" are same]
Loop through a map "orderDetail"
Alternative
[if total price of an order > $25,000]
return a buffer appending a customer name and order number
return OrdersDAO
return DAOFactory
return showCustomersOrder()
close Connection

Figure 4. Sequence diagram of the third requirement of set H

As some classes are shared between the requirements, the basic design of all 3 sequence diagrams is the same: user runs the main class, main class requests DAOFactory an show methods, DAOFactory requests DAOs, DAO classes then extend the DBConnection and request for the data, once they receive the data, the DAOs then manipulate data using getters from simple classes in different ways implementing the show methods. Then the main receives returned DAO from DAOFactory and show method for display. Lastly, once the main receives what it has requested, it closes the database connection.

Talking about differences, in the 1st requirement (figure2), ProductsDAO requests for each product line in the class Products using a for-loop and separate different product line in an ArrayList created. Then, it appends individual products and its information in a buffer. Lastly, it requests Products to format it in toString(). In the 2nd requirement (figure3), PaymentsDAO loops through Payments and gets date to assigned local variable. Using a mapping method, the DAO then checks if there are duplicate dates and accumulate the payment amount accordingly.

In the 3rd requirement (figure4), OrdersDAO loops through Orderdetails and using a mapping method it sums the price of the same order number. Next, it loops through Orders and Customers and turn them in ArrayLists. Then OrdersDAO loops through the customers ArrayList, then orders ArrayList, and then through orderdetails map according to certain conditions.

The test classes test DBConnection, ProductsDAO, PaymentsDAO, and OrdersDAO. They are made using JUnit4. They uses assertEquals method comparing each DAOs' method to the actual SQL database which uses MySQL statements such has INNER JOIN, SUM, UNION, and WHERE.