**UCN dmai0916**

**Group 2**



Mirjana Erceg

Tamas Kalapacs

Zahro-Madalina Khaji

Sangey Lama

Andreas Richardsen

# Table of Contents

[Table of Contents 2](#_Toc478719815)

[Mockup 3](#_Toc478719816)

[Fully dressed Use Cases 4](#_Toc478719817)

[Domain model 6](#_Toc478719818)

[Patterns and associations 6](#_Toc478719819)

[System sequence diagram – Sale order 7](#_Toc478719820)

[Interaction diagram 9](#_Toc478719821)

[System tests 10](#_Toc478719822)

[Scenario table 10](#_Toc478719823)

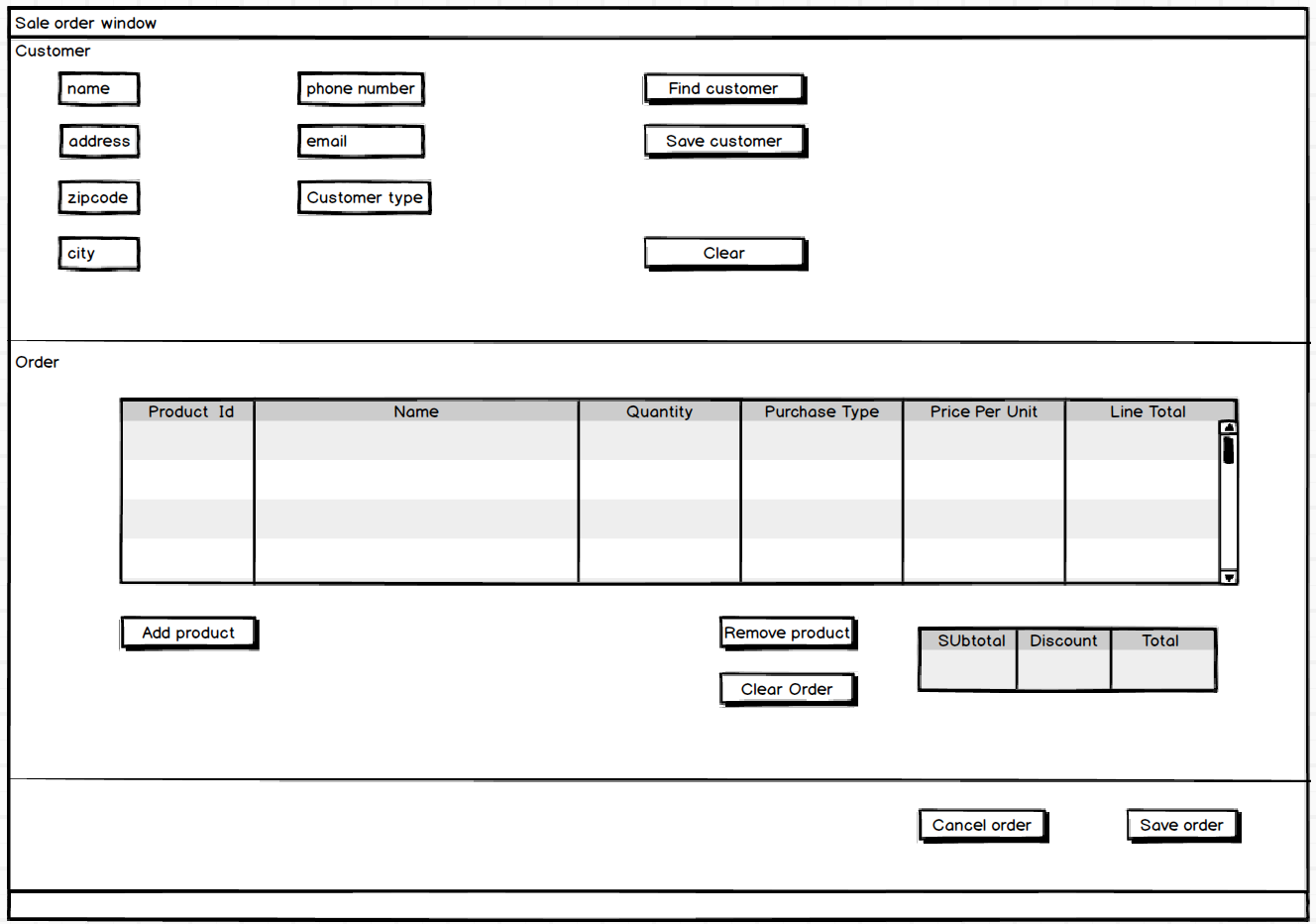
[Valid/Invalid table 10](#_Toc478719824)

[Real values table 10](#_Toc478719825)

[Code example for inserting a sale order into the database. 11](#_Toc478719826)

[Code standard 12](#_Toc478719827)

# Mockup



# Fully dressed Use Cases

|  |  |  |
| --- | --- | --- |
| Use Case Name | Process Sale Order | |
| Actors | Sales assistant | |
| Pre-condition | Employee is logged into the system, the system has a stable connection to the database | |
| Post-condition | Successfully created sale order | |
| Flow of events | Actor | System response |
|  | 1. Customer places an order through phone or email. |  |
|  | 2. Sales assistant starts a new sale. | 3. System creates a new blank sale. |
|  | 4. Sales assistant inputs a product id and quantity. | 5. System returns product details and subtotal. |
|  | 6. Repeat steps 4-5 as necessary |  |
|  | 7. Sales assistant inputs customer id | 8. System returns customer details and links the customer to the sale |
|  | 9. Sales assistant either tells the customer over the phone the total and gets confirmation or sends a confirmation email with the sale information. |  |
|  | 10.Sale assistant prints out an invoice and the delivery note and attaches it to the order. |  |

**Use case:** Process Sale Order  
**Scope:** Database Sale Order Processing System  
**Level:** User goals  
**Primary actor:** Sales Assistant  
**Preconditions:** Sales Assistant is logged into the system, an order has been placed by a customer.  
**Success Guarantee:** A Sale order with all the desired products has been created and saved. Discounts are applied according to customer’s membership. Stock and inventory are updated appropriately. An invoice is generated and printed out to be sent to the customer.  
**Main success scenario:**

1. Sales assistant creates a new Sale Order.
2. System displays a new blank Sale Order.
3. Sales assistant enters the customer identifier.
4. System displays customer information, name, address, city, zipcode and discount (if any).
5. Sales assistant enters the product identifier and quantity.
6. System displays sale line item with the item description, price, and sub-total. Prices are calculated based on a set of price rules.  
   *Sales assistant repeats steps 5-6 until order is filled.*
7. Sales assistant checks the order is correct and confirms the order.
8. System records the completed sale order, generates an invoice for the customer, and updates the stock accordingly.

**Extensions**

\*a. At any time customer requests to cancel the order.

1. Sales assistant cancels the order.
2. System does not record any information and exits.

3.a. Customer is not in the database.

1. Sales assistant enters the customer details manually.
2. Continue from main success scenario 5.

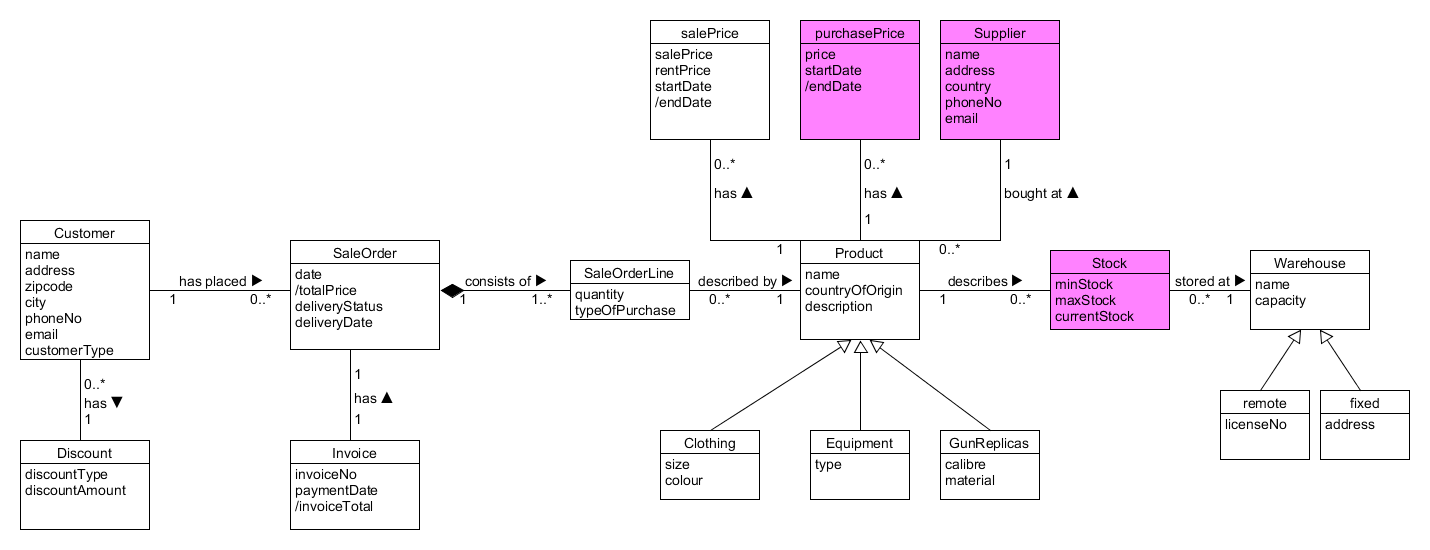
5.a. Product is not found by the system.

1. Sales assistant gives the option to the customer to continue or cancel the order.
   1. Customer tells sale assistant to cancel the order.
      1. Sale assistant cancels the order.
      2. System does not record any information, product stock remains unchanged, and system exits.
   2. Customer tells sale assistant to continue the order without the product.
      1. Sale assistant continues from main success scenario 5 without the missing product.

5.b. Insufficient quantity of product to fill order.

1. System creates a warning message display insufficient stock.
2. Sales assistant gives the options to the customer to continue with a new quantity or to remove the product from the order or cancel the whole order.
   1. Customer tells the sales assistant to change the quantity of the product.
      1. Sales assistant changes the quantity of the product.
      2. System displays the updated sale line item.
      3. Continue from main success scenario 5.
   2. Customer tells the sales assistant to remove the product from the order.
      1. Sale assistant removes the sale line item with the product.
      2. System displays the updated sale order without the product.
      3. Continue from main success scenario 5.
   3. Customer tells the sales assistant to cancel the whole order.
      1. Sales assistant cancels the order.
      2. System does not record any information, product stock remains unchanged and system exits.

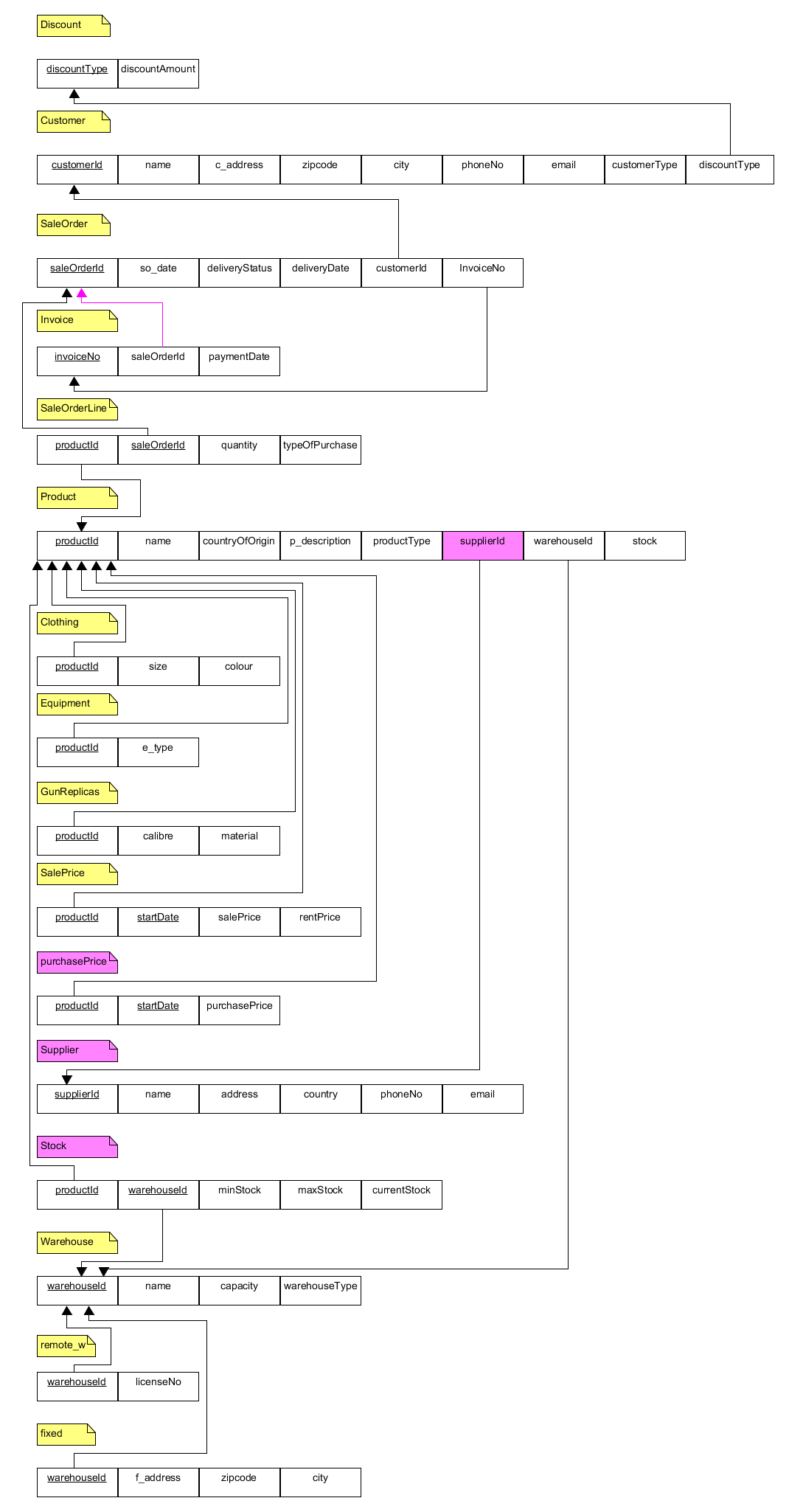
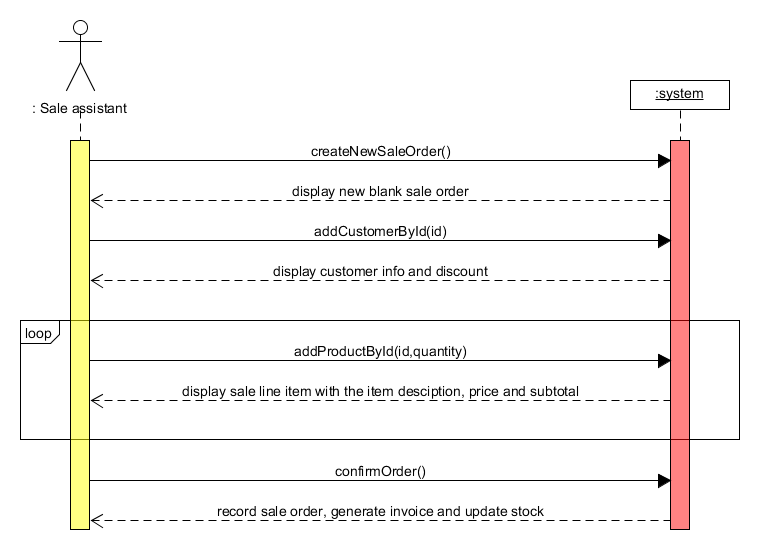
# Domain model



## Patterns and associations

In the domain model for the creation of Sales an agreement pattern is used. SaleOrderLine has a composite aggregation to SaleOrder which means if SaleOrder is deleted then SaleOrderLine is deleted with it. The hierarchy pattern means that the Product has fields that are common to Clothing, Equipment and GunReplicas and which they will inherit. The same goes for Warehouse and Remote and fixed. The classes PurchasePrice, Supplier and Stock will not be implemented in this iteration because of lack of time.

# System sequence diagram – Sale order

Relational model

For the transformation of the domain model to a relational model we use the 3NF.

SaleOrderLine has a composite primary key which guarantees its uniqueness.

We decided not to put everything on product and split up the product\_type

# Interaction diagram

# System tests

## Scenario table

|  |  |  |
| --- | --- | --- |
| Scenario name | Starting flow | Alternate flow |
| Scenario 1 – Successful creation of Sale order | Basic flow |  |
| Scenario 2 – Invalid customer id | Basic flow to step 2 | Extension 3a |
| Scenario 3 – Invalid product id | Basic flow to step 4 | Extension 5a |
| Scenario 4 - Insufficient quantity of product | Basic flow to step 4 | Extension 5b |

## Valid/Invalid table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test case Id | Scenario | Customer id is valid | Product id is valid | Quantity is valid | Expected result |
| 1 | Successful creation of Sale order | V | V | V | Sale order was created and recorded and an invoice was printed |
| 2 | Invalid customer id | I | N/A | N/A | Customer is not added to sale order |
| 3 | Invalid product id | V | I | N/A | Product is not added to sale order |
| 4 | Invalid product quantity | V | V | I | Displays stock error |

## Real values table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test case Id | Scenario | Customer id is valid | Product id is valid | Quantity is valid | Expected result |
| 1 | Successful creation of Sale order | 1 | 1 | 1 | Sale order was created and recorded and an invoice was printed |
| 2 | Invalid customer id | 333 | N/A | N/A | Customer is not added to sale order |
| 3 | Invalid product id | 1 | 999 | N/A | Product is not added to sale order |
| 4 | Invalid product quantity | 1 | 1 | 1000 | Displays stock error |

# Code example for inserting a sale order into the database.

Here we have our most important method for this use case the insertion of a sale order into our database.

The method parameters take the a SaleOrder object and we use that to try an insertiong.

In the beginning of the try block we establish a connection and start out transaction (setting autocommit to false)

Next we create an invoice stub to insert into the data first and then we associate the stub to our saleOrder object (so we can insert our invoiceNo value into our database.

After the invoice has been inserted we insert the SaleOrder object using a PreparedStatement to take the attributes from our SaleOrder object and then execute an update to the database.

Finally we call the insertSaleOrderLine method on every saleOrderLine element in our ArrayList of SaleOrderLines.

If all the methods and inserts complete without error, then we commit the transaction. However if an exception is thrown our catch block will rollback the entire transaction.

# Code standard

The names of the classes are nouns which start with a capital letter and they are simple and descriptive.

The package names are descriptive. The methods have names which start with a lowercase letter and every internal word starts with an upper-case letter. Our project contains a bunch of blank lines in order to improve its readability. A lot of comments are to be found in the project so as to be easier to understand. We have decided to use one declaration per line also because it improves the readability of the code. We have only imported the packages that is needed and not used any wildcards in order to make sure it is compatible with future java versions, in case its implemented a class in one of the packages we use that has the same name as one of our classes and so anyone that comes later to read the code will know what we used.

Revision number: 44

Database: <https://kraka.ucn.dk/svn/dmai0916_2Sem_2>