Persistens

Professionshøjskolen UCN
IT-Uddannelserne
Datamatiker
Gruppe 4

Navn (github navn)

Nicolai Mosgaard Jensen (Nicolai73)

Lucas Østergaard (Lucas Østergaard UCN)

Sebastian Abildgaard (Sebedusen)

Mathias Sørensen (ucnmathiassorensen)

Óli Jákup Finnsson (OJFinnsson)

DATO

13/10-2025

Brugte Server

hildur.ucn.dk

Database navn

DMA-CSD-V251_10647364

commit hash

02f4358

Resume

Vi er blevet stillet til opgave for at fremstille et system for Wastern Style Ltd. for at håndtere produkter, oprette/tilføje customer til oprettede order. Dette har vi opnået igennem brug af analytiske metoder samt et brug af UML og DAO. Efter udvikling af disse modeller som øger effektiviteten af de manuelle processer som Western Style Ltd. foretager sig.
Og for at lagre af data har vi stiftet bekendtskab til databaser og SQL.

Indholdsfortegnelse

Contents

Introduktion

Dansk detailvirksomhed, Western Style Ltd, grundlagt i 1994 af Hanne- og Børge Pedersen. Virksomheden har specialiseret sig i salg af westerninspireret tøj og tilbehør og driver en butik i Viborg. Western Style Deltage Hyppigt i markeder, festivaler og koncerter for at promovere deres produkter og nå ud til en bredere kundekreds.

Igennem årene er virksomheden vokset og har udvidet sit sortiment samt leverandørnetværk til både USA, Ungarn og Polen. Salget foregår i dag primært til danske kunder og turister, og ordrer håndteres manuelt via telefon og mail. Lagerstyringen udføres også manuelt, hvilket medfører udfordringer i forhold til overblik, effektivitet og rettidige genbestillinger.

For at imødekomme disse udfordringer har Western Style ønsker om at optimere deres IT-system. Formålet er at udvikle en mere automatiseret og integreret løsning, der kan understøtte lagerstyring, ordrebehandling og fremtidig ekspansion. Et opdateret IT-system vil give virksomheden bedre kontrol over lagerbeholdningen, reducere manuelle fejl og skabe et solidt grundlag for videre vækst – både nationalt og internationalt.

Problem statement

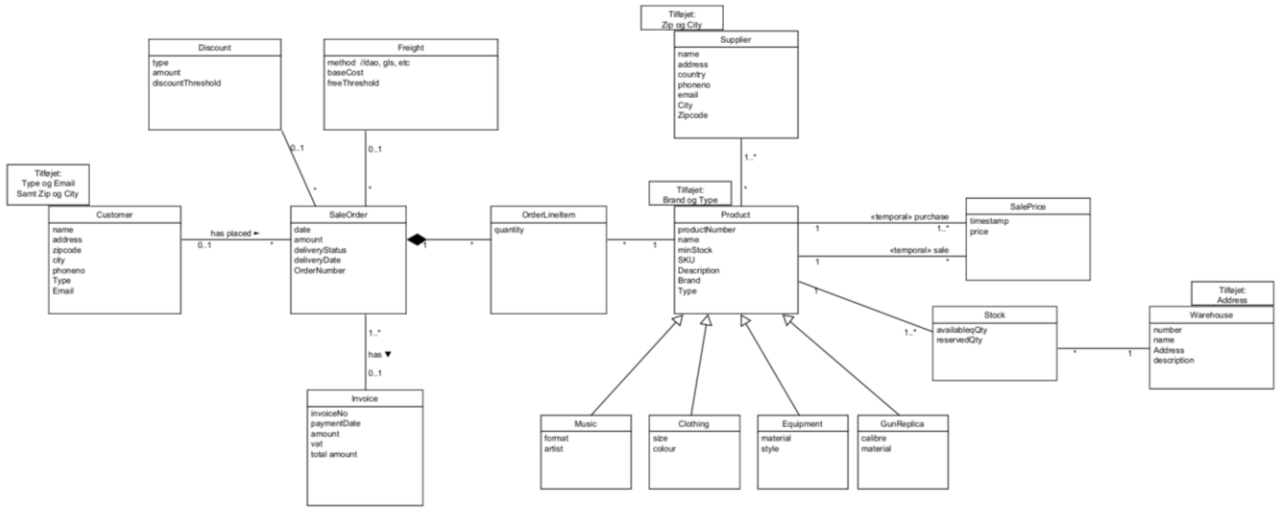
- Hvordan kan man, udvikle et tilfredsstillende program, som opnår alle de ønsker og behov, der er sat af Western Style Ltd., herunder automatisering af lagerstyring, optimering af ordrebehandling og understøttelse af fremtidig vækst?

Fully dressed Use-case

Use Case Name:	Place Order	
Primary Actor:	Employee	
Preconditions:	Medarbejderen har modtaget en ordre. Medarbejderen har fundet kundens data i databasen	
Postconditions:	Ordren er færdigregistreret.	
Frequency:	0 - 50 Gange om dagen.	
Flow of Events:	Actor - Employee	System -
	1. Medarbejderen placerer en ny ordre i systemet med Customer-type data.	2. Systemet laver en ny ordre.
	3. Medarbejderen indsætter produktdata.	4. Systemet søger efter ordrens indhold i Warehouse og returnere produktet og reservere det.
	5. 3 – 4 gentages.	6. Ordren bliver tildelt Discount
Alternative Flow:	1A. Customer-type eksisterer ikke i systemet. Ræk ud til kunde. 4A. Systemet kan ikke finde produktet i Warehouse. Det returnerer at produktet er invalidt.	

Use casen Place Order beskriver, hvordan en medarbejder hos Western Style Ltd. registrerer en ny ordre i systemet, efter at en kunde har placeret en bestilling via telefon eller e-mail. Formålet er at sikre en korrekt og effektiv håndtering af kunde- og produktdata, så varerne kan reserveres og klargøres til levering eller afhentning. Når medarbejderen modtager ordren, starter processen med at finde kundens oplysninger i databasen. Hvis kunden ikke allerede eksisterer, skal medarbejderen oprette en ny kundepost. Herefter oprettes en ny ordre, og medarbejderen registrerer de ønskede produkter i systemet. Systemet søger derefter i lagerdatabasen (Warehouse) for at kontrollere, om produkterne er på lager, og reserverer dem midlertidigt, så de ikke kan bestilles af andre.
For hver vare, der tilføjes, opdaterer systemet lagerbeholdningen og sikrer, at der tages højde for forskellige lagre, herunder det mobile lager, som bruges til festivaler og markeder. Når alle produkter er registreret, vurderer systemet, om kunden opfylder betingelserne for rabat eller gratis levering — f.eks. hvis et klubkøb overstiger 1.500 DKK får man rabat, eller en privatkunde handler for mere end 2.500 DKK får man gratis levering.
Når alle oplysninger er registreret og kontrolleret, markeres ordren som færdig, og systemet genererer en bekræftelse, der kan bruges til fakturering og afsendelse af varer. Er findes to alternativ forløb i use casen. Det første (1A) opstår, hvis kunden ikke findes i systemet. I dette tilfælde skal medarbejderen kontakte kunden for at indhente de nødvendige oplysninger og eventuelt oprette kunden i databasen. Det andet alternativ flow (4A) opstår, hvis systemet ikke kan finde et produkt i lagerdatabasen. I så flad informeres medarbejderen om, at produktet er ugyldigt eller ikke på lager, hvorefter produktet må fjernes fra ordren eller erstattes af et andet. Når alle trin er gennemført uden fejl, er ordren fuldt registreret og gemt i systemet.

Domain model



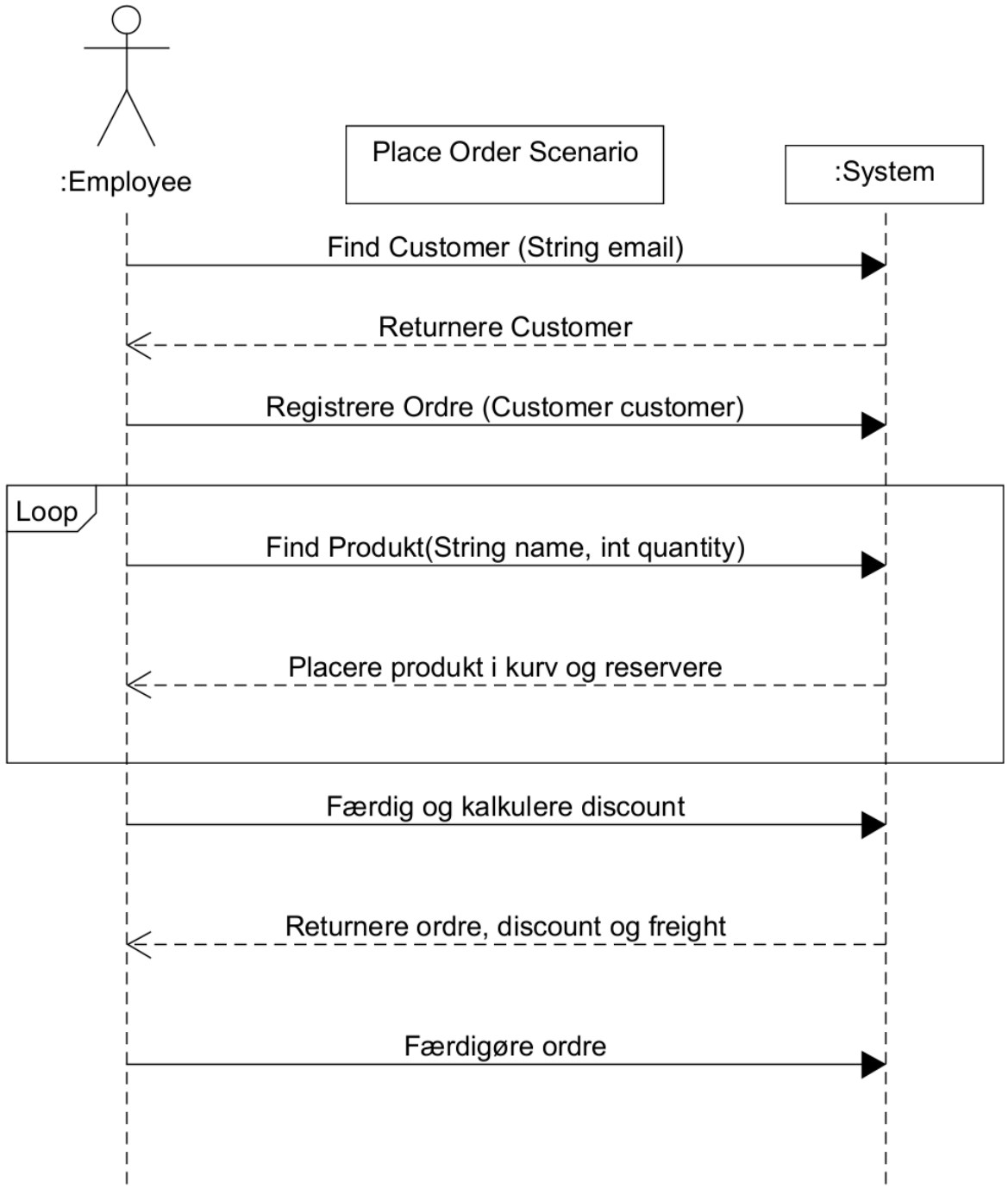
Domænemodellen beskriver, hvordan kunder, produkter, lager og prissætning spiller sammen i forbindelse med et salg. En kunde registreres i Customer med navn, adresse og kontaktoplysninger, og kan placere nul eller flere SaleOrder. En ordre er ordrehovedet med dato, beløb, ordrenummer, leveringsstatus og evt. leveringsdato. Hver ordre består af en eller flere OrderLineItem, hvor hver linje angiver den bestilte mængde af et bestemt Product. Produktkataloget er samlet i Product, som bærer varige egenskaber som varenummer, navn, SKU, beskrivelse, brand og type. Produktet specialiseres i underklasserne Music, Clothing, Equipment og GunReplica, der tilfører domænespecifikke felter (fx format og kunstner for Music, størrelse og farve for Clothing, samt kaliber og materiale for GunReplica). Produkter kan have en eller flere leverandører via Supplier, som beskriver leverandørens navn, adresse og kontaktdata.

Lagerdimensionen modelleres eksplicit for at adskille varens identitet fra dens fysiske beholdning. Warehouse repræsenterer et lagersted (fx butik eller mobilt lager) med navn og adresse. Forholdet mellem lager og produkt udtrykkes i Stock, som angiver availableQty og reservedQty for hvert produkt på hvert lager. Dermed kan systemet opgøre disponibel mængde som available minus reserved og sikre, at reservationer ved ordreoprettelse ikke overskrider den faktiske beholdning. Strukturen tillader, at samme produkt findes på flere lagre, og at et lager rummer mange produkter, uden at duplikere produktdata.

Prissætning og tidsafhængige priser håndteres af SalePrice, der gemmer pris og timestamp som en temporal historik. Når en ordre afsluttes, afgøres den gældende salgspris ud fra produktets prislinjer på tidspunktet for købet. Forretningsregler for rabat og fragt er modelleret som selvstændige begreber: Discount knyttes 0..1 til ordren og beskriver type, sats og en beløbsgrænse for udløsning af rabat, mens Freight tilsvarende beskriver leveringsmetode, basisomkostning og evt. fri-fragt-grænse. Disse to objekter gør det muligt at ændre satser og tærskler uden at påvirke kerneobjekterne for ordre og lager. Betalings- og afregningssporet opsummeres i Invoice, som udstedes 0..1 pr. ordre og indeholder fakturanummer, betalingsmetode, beløb og total. Fakturaen er afledt af ordren og dens linjer, den valgte pris på købstidspunktet, samt de regler for rabat og fragt, der gælder i situationen. Samlet set sikrer modellen klar ansvarfordeling: Customer ejer sine ordrer, SaleOrder samler økonomien på ordreniveau, OrderLineItem binder mængde og produkt, Product og dets specialiseringer bærer varige vareegenskaber, Stock og Warehouse håndterer mængder pr. lager, mens SalePrice, Discount og Freight styrer pris, rabat og fragt uden at blande sig i lagerlogikken. Denne opdeling understøtter sporbarhed, konsistens og en direkte vej til et normaliseret databaseskema.

System sequence diagram

Efter en fully dressed use-case, samt en domain model, kan et system sequence diagram (SSD) fremstilles. En SSD bruges til at få et bedre overblik om hvordan systemet interagere med eksterne aktører.



Den fysiske aktør, den ansatte, starter processen med at sende en inquery om at finde en specifik kunde via kundens e-mail. Systemet returnerer kundens info. Nu når aktøren har dette info fra systemet kan de starte deres ordre.

Efter at ordren er skabt kan aktøren tilføje alle de nødvendige produkter til ordren. Siden denne proces skal gentages en varierende mængde gange repræsentere vi det med en loopboks.
Når alle produkter er tilføjet til ordren sender aktøren en besked til systemet. Systemet udregner herefter pris, rabat, og fragt. Når systemet er færdig med udregningerne sender den dem tilbage til aktøren, og aktøren kan slutte processen.

Operations contract

Efter at SDD'en er skabt, kan man tage og analysere den for at finde de forskellige funktioner som indgår i use-casen. Efter at alle funktionerne er fundet skal der findes post- og preconditions for hver funktion, for at finde alle krav som er nødvendige for at use-casen kan ske.

Operation: Find Customer (String email)
Use Case: Place Order
Preconditions:

- Employee skal have en customer's email

Postconditions:

- Vi får en customer reference tilbage

Operation: Registrere Ordre (Customer customer)
Use Case: Place Order
Preconditions:

- Customer customer = Customer.getCustomer

Postconditions:

- En ny instance af Order order er dannet
- Order bliver associeret med customer

Operation: Find Produkt(String name, int quantity)
Use Case: Place Order
Preconditions:

- En ny instance af Orderline ol dannes
- ol associeret med Produkt og order
- En Order instance skal være skabt

- Et customer objekt skal være tilknyttet
- quantity skal være valid
- Produktet skal eksistere i systemet

Postconditions:

- Produktet er valid

Operation: Færdig og kalkulere discount
Use Case: Place Order
Preconditions:

- Loopet af “Find Produkt” og ”placere produkt-” skal være stopped

Postconditions:

- Ordre er færdig

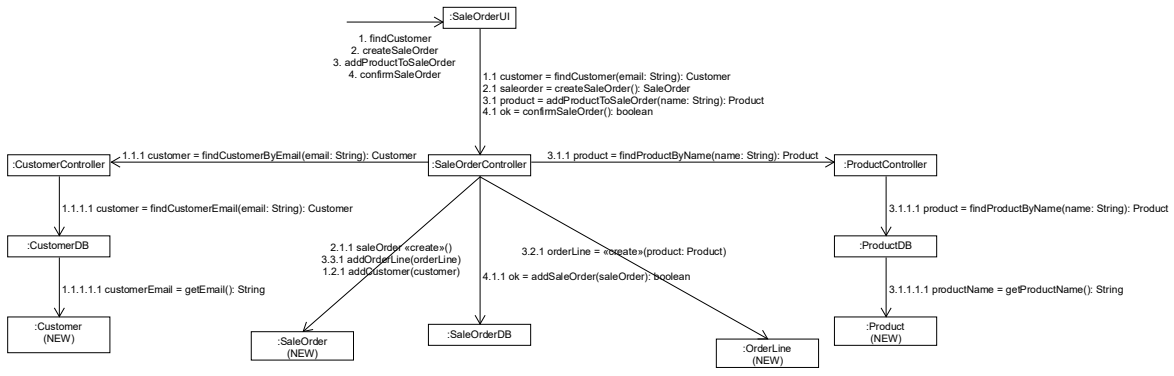
Operation: Færdigøre ordre
Use Case: Place Order
Preconditions:

- Skal modtag bekræftelse.

Postconditions:

Interactions diagram

Med et samlet brug af den skabte fully dressed use-case, domain model, og SDD, kan et interaktionsdiagram laves. Dette diagram laves til at få en visuel ide om hvordan de forskellige operationer interagere med de forskellige dele af systemet.



Ved at observere interaktionsdiagrammet kan det ses at denne use-case har 4 forskellige operationer som den skal foretage sig. Ved at yderligere kigge på interaktionsdiagrammet kan det ses hvilke dele af programmet de forskellige operationer interagere med.
Den første operation, findCustomer, Sender en kommando, findCustomer, til SaleOrderController, hvor at den yderligere går ned igennem CustomerController, CustomerDB, og Customer. Efter at den har hentet alt det den har brug for fra Customer, sender den det videre til SaleOrder i form af en addCustomer funktion.
Den anden operation er createSaleOrder. Denne funktion skaber vores SaleOrder.
Den tredje funktion er det loop som kan ses i det forrige SSD. Denne funktioner gå ned og henter alle de produkter som der skal bruges. Dette gøres med findProductBtName funktionen, som returnerer produktet i formen af en string. Efter at den har fundet et produkt bliver det givne produkt tilføjet til OrderLine. Her ligger alle produkter indtil at alle produkter er blevet fundet. Når alle produkterne som er ønsket er fundet og tilføjet til OrderLine, bliver de tilføjet til SaleOrder.
Den sidste operation er confirmSaleOrder. Denne funktion gør ikke mere and at afslutte use-casen, og sende den videre.

Test cases

Formålet med disse test cases er at validere, at use casen *Place Order* fungerer som forventet i både normal- og fejlscenarier.
Testene dækker de flows, der er beskrevet i den *Fully Dressed Use Case*:

- **Basic Flow** (ordre oprettes korrekt)
- **Alternative Flow 1A** (kunden findes ikke)
- **Alternative Flow 4A** (produkt ikke på lager)

Systemet skal korrekt håndtere ordreoprettelse, produktreservation, rabat- og fragtregler samt fejlscenarier.

Use Case Scenarios

Scenario ID	Scenario Description	Flows
1	Successful place order	Basic Flow
2	Customer eksisterer ikke	Basic Flow + 1A
3	Produkter findes ikke i Warehouse	Basic Flow + 4A

Scenario Matrix

Scenario 1 Successful place order	Basic Flow		
Scenario 2 Customer eksisterer ikke	Basic Flow	1A	
Scenario 3 Produkter findes ikke i Warehouse	Basic Flow	4A	

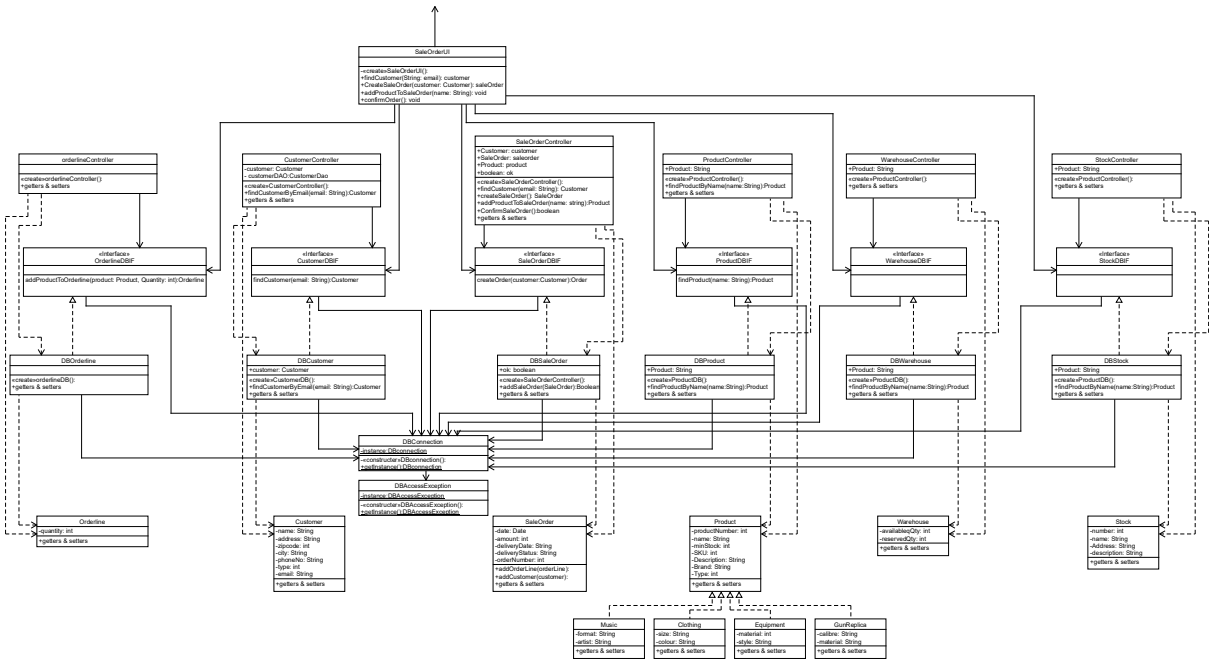
Test Case Matrix

Test ID	Scenario / Condition	Customer Email	Customer Type	Product Name	Product Type	Warehouse Quantity	Discount Type
1	Succesfully Placed Order	V	V	V	V	V	N/A
2	Ikke nok varer på warehus	V	V	V	V	I	N/A
3	Product Type findes ikke	V	V	V	I	N/A	N/A
4	Product name findes ikke	V	V	I	N/A	N/A	N/A
5	Customer typen findes ikke	V	I	N/A	N/A	N/A	N/A
6	Email findes ikke	I	N/A	N/A	N/A	N/A	N/A

Disse test cases sikrer, at Place Order-processen fungerer korrekt i både normal- og fejlscenarier, samt at systemet håndterer rabat-, fragt- og lagerregler i overensstemmelse med forretningskravene fra casen Western Style Ltd.

Design class diagram

Med udgangspunkt i interaktionsdiagrammet, og domain modellen kan det sidste diagram laves inden at meningsfuld kode kan blive skrevet.
Et Design class diagram (DCD) Bruges til at få et overblik over hvilke klasser som programmet kommer til at skal bruge, hvilke funktioner og operationer som hver klasse skal indeholde, og hvordan de interagere med hinanden.



Dette DCD er blevet skabt med data access object (DAO) i tankerne. Hvor de forskellige klasser er delt op i tre dele. Fra bunden op er de delt ind i model, database og controller. Det kan ses at der er forskellige typer af pile som peger imellem de forskellige klasser. Den Fuldt optrukket linje indikerer at den klasser som peger på en anden klasse, bruger den anden klasse. Hver gang der er en fuldt optrukket linje, kan der huskes at skabe en konstruktør i den brugte klasse. En Stiplet linje med pil uden bagside indikerer at en klasse tager info fra en klasse, eller ændrer i den klasse. Dette kan ses imellem hver database og den korresponderende model. En stiplet linje med en hul pil indikerer at den klasse som peger på en anden klasse, inheritor fra den anden klasse.

SQL scripts

Creation of tables in database

```
CREATE TABLE Customer (
ID INT IDENTITY (1,1) PRIMARY KEY NOT NULL,
Fname varchar(50) NOT NULL,
Lname varchar(50) NOT NULL,
Address varchar(50) NOT NULL,
PhoneNo varchar(50) NOT NULL,
Email varchar(50) NOT NULL,
Type INT NOT NULL,
Zipcode INT NOT NULL FOREIGN KEY REFERENCES ZipTable(Zip)
)

CREATE TABLE [Product] (
ID INT IDENTITY (1,1) PRIMARY KEY NOT NULL,
ProductNumber INT UNIQUE NOT NULL,
ProductName varchar(50) NOT NULL,
MinStock int,
Description varchar (500),
SKU INT NOT NULL,
Type INT
)
```

denne SQL Querry, er blevet brugt til at skabe tables i vores database. Ud fra vores Domain Model, hvoraf der blev skabt et Relational Schema.
Der er Foreign Keys til andre tables som de gør brug af. Det vil dermed blive brugt sammen med Prepared Statements inde i vores IDE for Java.
ID i disse tables, er Auto-Generated keys. Dvs at hver gang der for eksempel bliver skabt en ny Customer. Vil den få tildelt sin egen unikke Primary ID Key.
CREATE TABLE er opsat til at blive skabt i den rigtige rækkefølge, så de påsatte Foreign Keys bliver tildelt til de rigtige Tables.

Insertion of data into database

```
insert into ZipTable (Zip, City)
VALUES (9000, 'Aalborg');

insert into ZipTable (Zip, City)
VALUES (9800, 'Sæby');

insert into ZipTable (Zip, City)
VALUES (9830, 'Taars');
```

Dette script indsætter værdier til den Primary Key, i dette tilfælde Zip i Tablet ZipTable.
det sikrer at der kun kan være den specifikke by til sit Primary Key nummer.
ZipTable er et join table, hvorved andre klasser der skal skal bruge en by referere til denne Zip Primary Key.

```
INSERT INTO Customer (Fname, Lname, Address, PhoneNo, Email, Type, Zipcode)
VALUES ('John', 'Doe', '123 Maple Street', '555-123-4567', 'john.doe@email.com', 1, 9000);

INSERT INTO Customer (Fname, Lname, Address, PhoneNo, Email, Type, Zipcode)
VALUES ('Sarah', 'Lee', '456 Oak Avenue', '555-987-6543', 'sarah.lee@email.com', 2, 9800);

INSERT INTO Customer (Fname, Lname, Address, PhoneNo, Email, Type, Zipcode)
VALUES ('Michael', 'Brown', '789 Pine Road', '555-321-9876', 'michael.brown@email.com', 1, 9830);
```

Her er endnu et insert, hvor der sættes data ind i Customer Table.
Der bliver tildelt de nødvendige varchar samt INT, til Type og Zipcode. Hvoraf Zipcode er en Foreign Key til ZipTable. For at skulle få fat i den anlagte City.

```
drop table GunReplicaTable

drop table EquipmentTable

drop table ClothingTable

drop table MusicTable

drop table Stock

drop table Warehouse

drop table SalePrice

drop table ProductSupplier

drop table OrderlineItem

drop table SaleOrder

drop table Discount

drop table Invoice

drop table Freight
```

Har derud over lavet et SQL Querry, i tilfælde af, at man skulle blive nød til at drop table. Og fjerne dem fra databasen.
De er sat op i den rigtige rækkefølge, for ikke at skulle gå i kampulage med Foreign key.

Code Standard

Når det kommer til at skrive kode er det med alles interrese i tankerne at alle standard bliver overholdt for skrevet kode. En kode standard skaber et grundlag for de forskellige konventioner som skal medfølge koden. Med Oracle's java code conventions som udgangspunkt er en række af kode standarder blevet udvalgt som specielt vigtige at overholde.

Navngivning

Når det kommer til at navngive forskellige ting er det kritisk er overholde en række af navngivnings konventioner. For dette kode er to navngivnings konventioner primært brug til at sørge for at det er overskueligt er læse den skreven kode.

Pascalcase

Når en klasse skal navngives bliver der taget brug af Pascalcase. Pascalcase diktere at det første bogstav i klassen navn er versal.

camelCase

Det anden navngivnings konvention handler om metode. camelCase diktere at det første bogstav i en metode skal skrives med småt, og hvis mere end et ord er inkluderet i navnet så skal alle efter det første ord skrives med stort

Formatering

Linebreak indryk

For at øge læseligheden af koden foretages der et indryk efter linebreak. Dette indryk er en afstand af hvad der svarer til 4 mellemrum. Dette indryk skal kun ske efter et keyword som if, for og lign.

Placering af tuborgklammer.

For at gøre det nemmere at se hvornår forskellige funktioner og operationer slutter bliver der sørget for at der sker et linebreak inden af der bliver placeret en tuborgklamme.

Code Examples

```
1  package tryMe;
2
3  import java.sql.SQLException;
7
8  public class TryMe {
9
10     public static void main(String[] args) throws SQLException, DataAccessException {
11
12     //  DBConnection db = new DBConnection().getInstance();
13
14         CustomerDAO cdao = new DBCustomer();
15         SaleOrderDAO sodao = new DBSaleOrder();
16
17         Customer customer = cdao.findByEmail("john.doe@email.com");
18
19         int id = customer.getId();
20
21         SaleOrder order = sodao.createOrder(id);
22         System.out.println("new order with customer of id " + order.getCustomerID());
23         System.out.println("Id " + order.getCustomerID() + " = " + customer.toReadableString());
24
25     }
26 }
27
```

Her i TryMe Classen initierer vi en vores order ved at finde Customer via en given email.
Koden skaber nye databaseobjekter gennem DAO'er (interfaces)
Med dette kan vi hente en Customer gennem interfacet ved at kalde på dens method (findByEmail) hvilket så returner en Customer
For at vi kan læse Customerns data kalder vi et method i Customer som konverterer alt dens data til en string, som vi kan læse.

Konklusion

Igennem arbejde med place order use-casen er der blevet designet et system som kan fuldføre ønskerne sat af Western Style Ltd.
Med primær brug af på UML og DAO er der blevet dannet et grundigt overblik af systemets struktur og funktionalitet.
Med dette er der blevet skabt et system som giver Western Style Ltd. mere kontrol over deres lager, mere effektiv ordrebehandling og arbejdsproces.