Workshop Persistence

University College Nordjylland

Datamatikeruddannelsen

Klasse: dmaa0216

Titel: Workshop Persistence

Versionskontrol-sti: https://github.com/mrurb/Workshop-persistans/invitations versionsnummer: 82

Projektdeltagere:

Andersen, Laurids

Urban, Anders

Sørensen, Daniel

Simonsen, Martin

Vejledere:

Henrik Kristian Ulrik Øllgaard

Lars Landberg Toftegaard

Afleveringsdato:

14/10-16

Use Case use cases

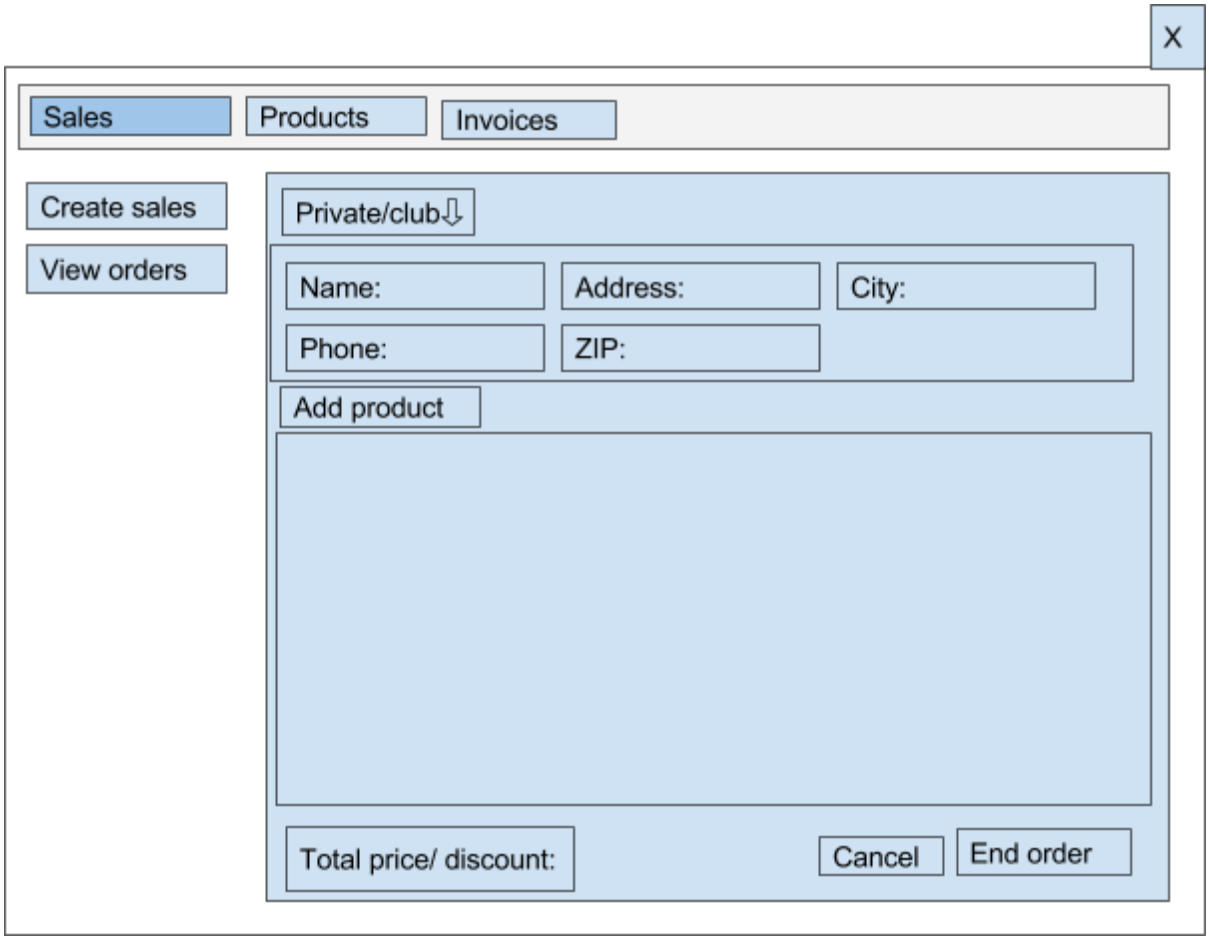
Use case: CRUD operations

Børge and Hanne acquires new products, and other employees are responsible for adding them to the database. The employees interact with the database and make basic Create, Read, Update and Delete (CRUD) operations.

Use case: Create Sales Order

Use Case	Create sales order	
Actors	Sales assistant	
Pre-conditions	Customer contacts with regards to an order	
Post-conditions	Products are packed and will be sent	
Frequency	Every sale	
Main Success scenario (flow of events)	Actor (action) sales assistant	
	1. Sales assistant creates sales order	2. System shows order
	3. Sales assistant adds products to order	4. System shows order with products.
	5. Sales assistant ends order	6. System shows invoice and gives option to print delivery note
	7. Sales assistant packs the order, delivery note attached	
alternative flows	3a. Product(s) not in stock. b. Order is cancelled.	

Mock-ups

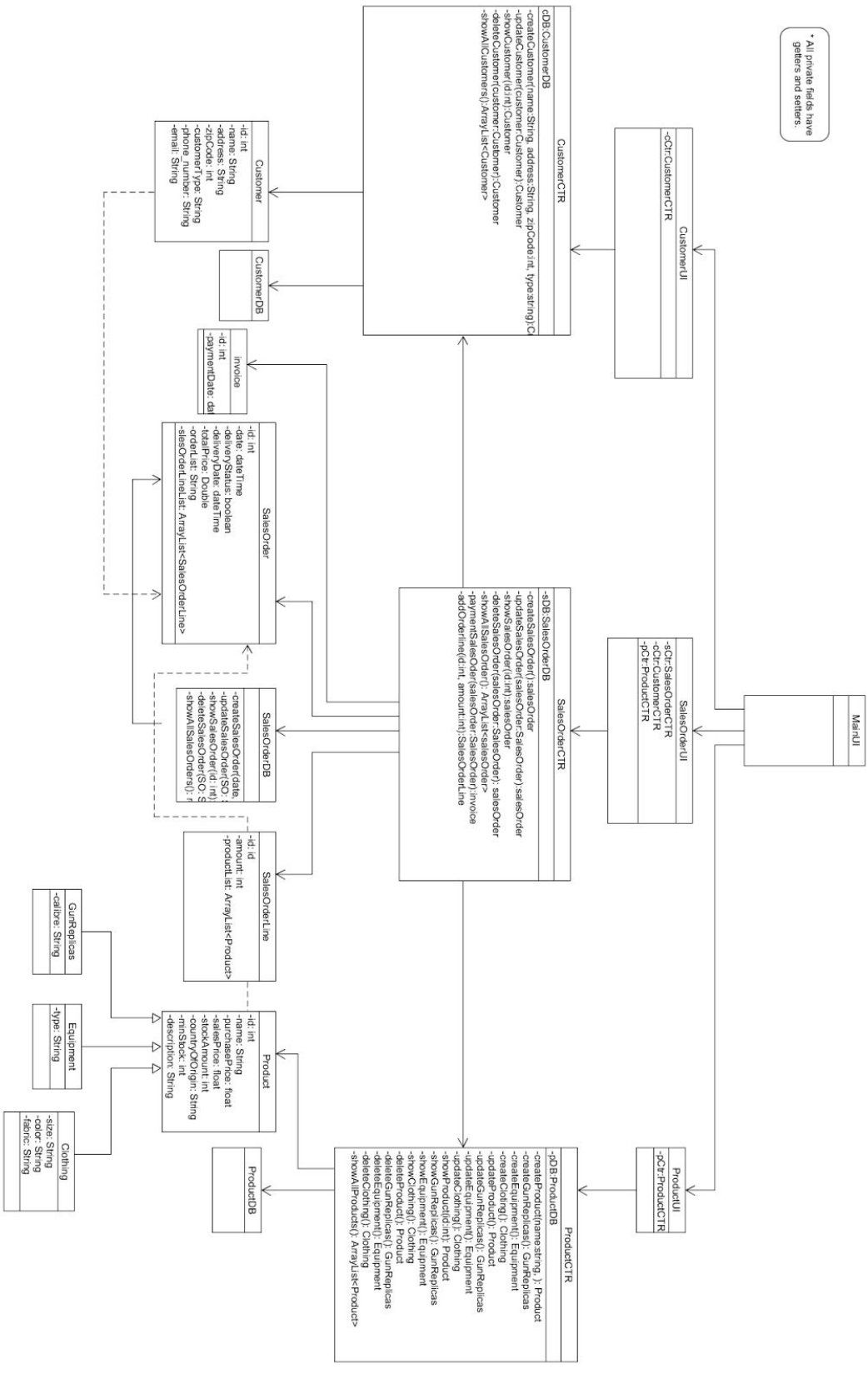


Vi vælger at designe programmet på følgende måde, da vi mener at en meget simpel opbygning er godt til et meget simpelt projekt, hvor der ikke er behov for unødvendige funktioner.

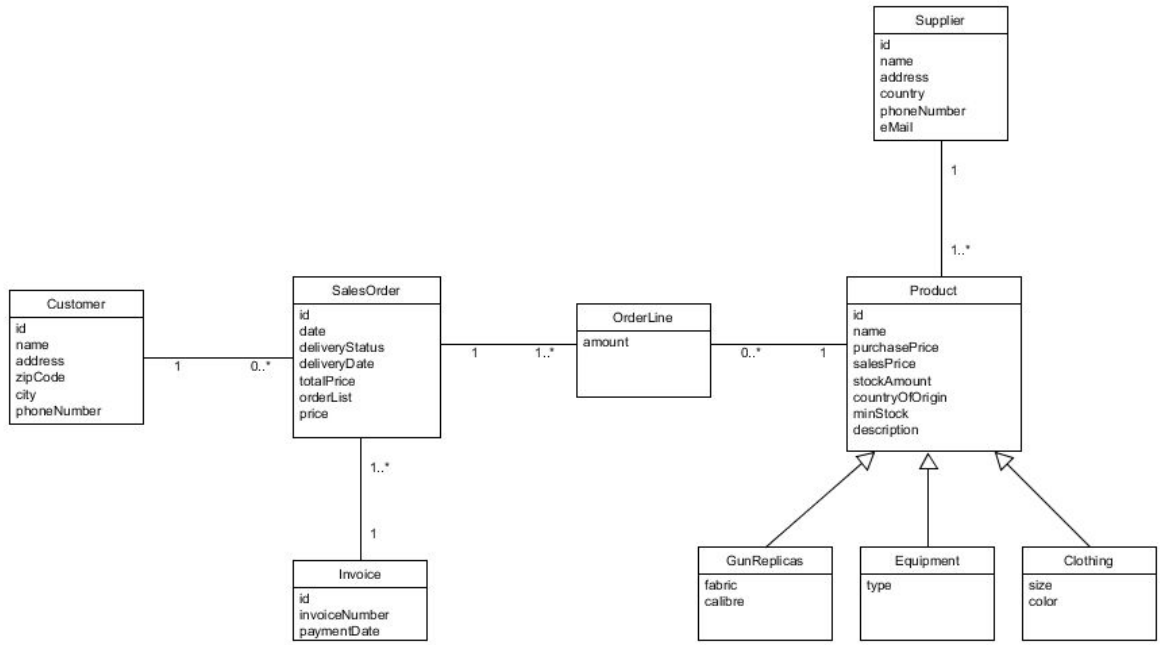
I udviklingsfasen, har der været genovervejelser i forhold til designet. Bl.a. startede programmet med at kunder ikke blev gemt, dette blev efterfølgende lavet om på, da det vil være en god tilføjelse til den givet usecase.

Models

Design Class Diagram

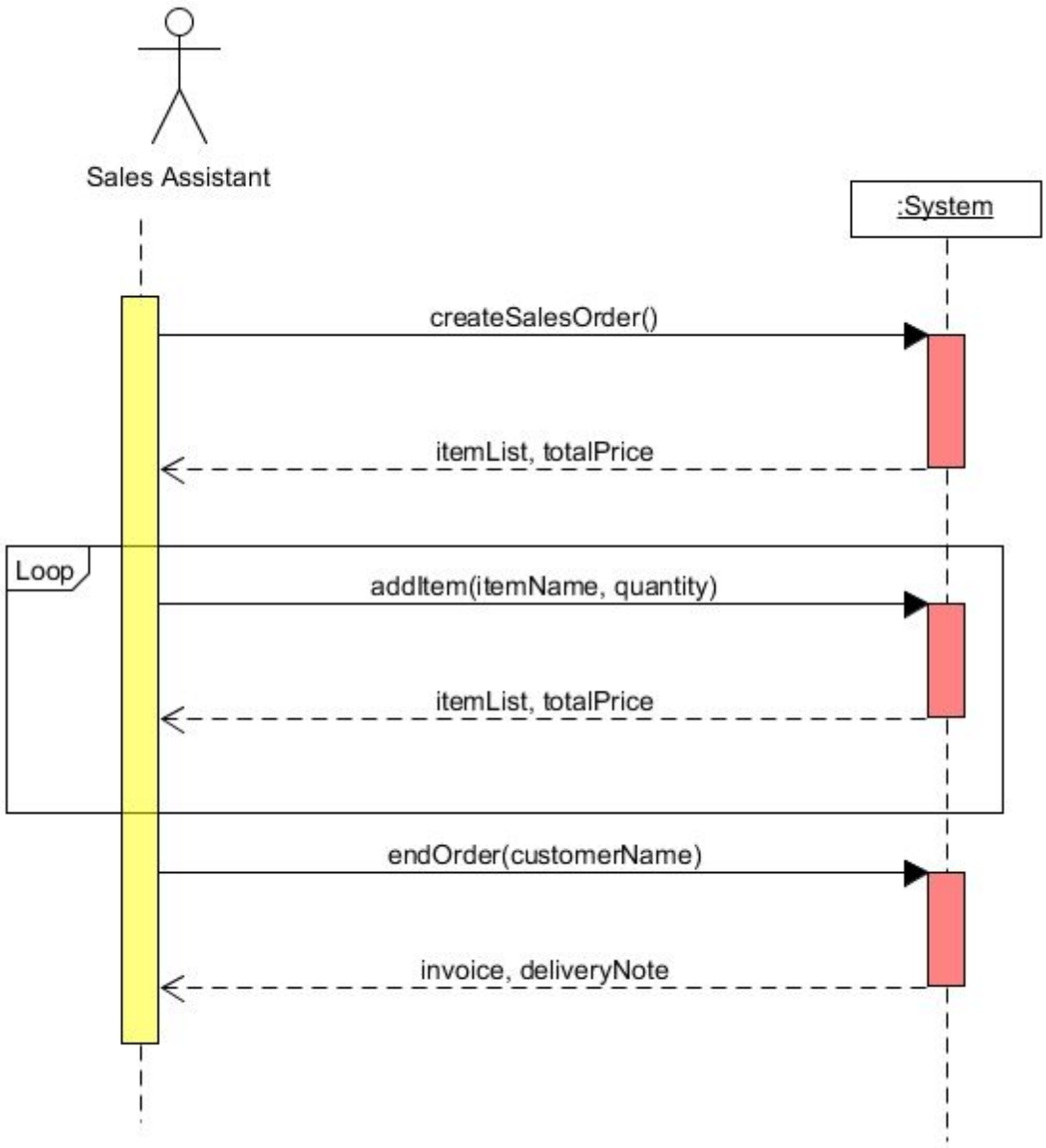


Domain Model



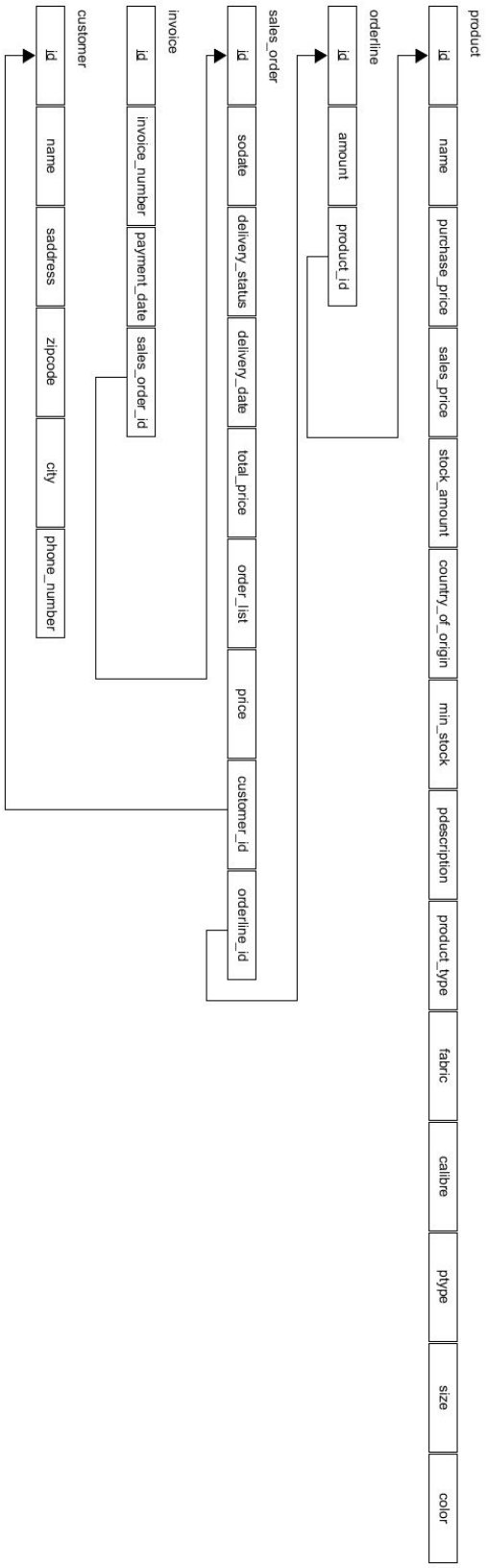
Sat klassen “OrderLine” ind mellem “SalesOrder” og “Product” for at holde styr på mængden af produkter der sælges gennem objektet “SalesOrder”.
Attribut “description” er sat på “Product” for at tillade en beskrivelse for alle “Products” børn.
Fravalgt “rentPrice” i klassen “Product”, fordi dette kraver et customer objekt som gemmes databasen.
Attribut “totalPrice” tilføjet Product som erstatning for “amount”, gemmer den rigtige pris(som ikke ændres).
Attribut “orderList” tilføjet Product for at undgå direkte reference til Product.
Attribut “id” tilføjet klasserne “Customer”, “SalesOrder”, “Invoice”, “Supplier”, og “Product”, dette skal bruges i forbindelse med databasestyringen.
Klassen “Supplier” er muligvis ikke interessant at arbejde videre med idet at det ikke er kritisk information for at kunne lave et salg. Til gengæld beholdes som en fremtidig udvikling, skulle det blive nødvendigt.

Sequence Diagram



Af systemsekvensdiagrammet skal læses at en servicemedarbejder(Sales Assistant) opretter en salgsordre ved interaktion med systemet(System) for use casen “Create Sales Order”. I det medarbejderen opretter en salgsordre, returnerer systemet først en tom list og en totalpris, herefter tilføjer medarbejderen varer til listen, hver gang dette sker opdateres listen og totalprisen for salget som vises til medarbejderen. Når der ikke ønskes flere varer at sætte på listen afslutter medarbejderen ordren og til sidst returnerer systemet faktura, leveringsseddel, og evt. mere.

Relation Database Schema



For at holde databasens opbygning simpel er attributterne fra superklassen Product smidt sammen i Products database skema; dette laver i sidste ende en mere generaliseret database klasse.
Der er tilføjet et ID kolonne til alle klasser i database skemaet, det er gjort for blandt andet at få mulighed for at have et garanteret unikt primær nøgle. På den måde sørger vi for at hvert objekt altid er unikt.
I tabellen *order_line* er der en fremmednøgle til *produkt* tabellen. Fremmednøglen peger på produkttabellens primære nøgle. Alle tabellers primære nøgle er kaldet id, og det gør dem alle identificerbare. Dette gør at der er en relation mellem de to tabeller, så man kan koble tabellerne, men stadig have dem overskuelig. Alle *order_line* tabellerne peger på *sales_order* tabellen fordi der er mange ordrelinjer til en enkelt salgsordre.

Manual for at åbne programmet

For at starte programmet, skal brugeren klikke på den vedhæftede .bat-fil. Derefter åbnes programmets startvindue, som er menuen *Sales UI*.
Under den første menu *CustomerUI* er det muligt at tilføje nye kunder, ved at trykke på tabben *Tilføj kunde* og derefter udfylde informationen og vælge om det er en privat kunde eller en klub man opretter, derefter skal knappen *Submit* trykkes på. Desuden er det også muligt at få vist alle kunder, ved at trykke på tabben *Vis alle kunder*. Knappen *Opdater* gør det muligt at opdatere listen af kunder, hvis der er blevet tilføjet en ny kunde imens, at man har kigget på listen. Alle kunder har en knap, *slet,* som er tænkt til at kunne slette de enkelte kunde. Grundet tidsmangel, har denne knap ikke nogen funktion på nuværende tidspunkt, men vil senere være en funktion som skal gøre det muligt at slette den pågældende kunde.
Fremgangsmåden er den samme for menuen *ProductUI,* det er her at man opretter et nyt produkt. Man udfylder den information som man ønsker at det pågældende produkt skal have og derefter trykker man på *submit.* Hvis man ønsker at se alle produkter trykker man på tabben *alle produkter. Det er muligt at trykke på knappen Slet*, dog gør den ikke noget, da der ikke var tid til at indføre denne funktion. Dette er dog en funktion som ville blive tilføjet senere.
Under menuen *Sales UI*, er det muligt at oprette et nyt salg. Først skal man skrive id på den kunde som man ønsker at oprette et nyt salg til, og derefter skriver man et id på det produkt som man ønsker at tilføje til ordren og antallet af dette produkt. Når alle produkterne er tilføjet til ordren, trykker man på *opret ordre.* Hvis man ønsker at se alle ordrer trykker man på tabben *Vis alle salgs ordrer* hvor man får en liste af salgsordre, hvor det er muligt at slette hver enkelt ordre, ved at trykke på *Slet*. Dette virker som ikke endnu, da der ikke har været tid til at implementere funktionen til denne knap.
Desuden er *invoice* designet og sat ind i databasens opbygning, dog er selve funktionen blevet afgrænset, grundet tidsmangel.

Beskrivelse af Test

Der har været anvendt JUnit til udførelsen af unit test for at sikre at resultaterne af enkelte metoder fremkommer som forventet.
JUnit-testen er en black-box test, der udelukkende ser på det forventede output versus det aktuelle output (som opnås ved at udføre metoden). Der er 3 testklasser; en til hver af de 3 create-metoder: createCustomer, createProduct og createSalesOrder. Der bliver i disse tests lavet det forventede output, og derefter bliver testen foretaget med en række parametre, og metoden bliver kørt. Det aktuelle output sammenlignes, og et resultat Sandt eller Falsk kommer tilbage (*Var de ens?*).
I en blackbox test ses der bort fra detaljerne, og man tester en større del af et program, for at sikre sig, at alt i denne del af programmet fungerer som forventet. Hvis resultatet ikke er som man forventer det, så må man antage, at der er en fejl i denne del af programmet (evt. en central metode).
Da størrelsen af projektet er lille og generelt begrænset, desuden grundet manglende erfaring, har det været mest hensigtsmæssigt at teste manuelt efter hver større ændring, så som tilføjelse af en metode eller en SQL query.

Beskrivelse af Udvalgt Kode

Herunder findes der et udpluk af kildekoden. Denne del stammer fra DBAccess laget, i SalesOrderDB klassen. Generelt, indeholder SalesOrderDB meget af den forretningskritiske databasekode, og behandler interaktion mellem programmet og databasen, primært *sales_order* tabellen (der er dog også en del fra *customer*, *produkt* og *order_line*, der er enten vedhæftet med en JOIN funktion eller hentet direkte).
Koden herunder behandler en "**createSalesOrder**" operation, hvor man opretter en salgsordre i databasen.
Først oprettes der et PreparedStatement objekt (som sættes til null). Så oprettes vores baseQuery (som er den SQL-streng vi sender til databasen, efter at den er blevet forberedt). Connection **'con'** oprettes og sættes til null. **'con'** er vores biblioteks-klasse, som vi anvender til at oprette forbindelse til databasen. Denne klasse har vi selv skrevet, men tager hovedsageligt udgangspunkt i klassen fra undervisningen (det er skrevet fra bunden, men tager udgangspunkt i dette format; dog med mindre rettelser).
'con' kalder bibliotekets singleton klasse og opretter forbindelse med "**getDBcon()**" metoden.
Derefter opsættes PreparedStatement klassen med parametrene **baseQuery** samt en Statement, som returnerer den genererede nøgle, når baseQuery bliver eksekveret. Den bliver nødvendig senere.
I linjerne efter bliver prepared statement opsat; alle spørgsmålstegnene fra baseQuery bliver erstattet med værdier, ved brug af metoder fra PreparedStatement; hvilket er med til at sikre, at typerne passer, og at der ikke opstår fejl (eller injections) med værdier som fx gåseøjn(").

Dette gør programmet stabilt, således at man ikke kan skabe problemer. Ville man fx indsætte nogle værdier fra GUI laget, men kom til at sende et enkelt sæt gåseøjne afsted, fx **Jens Pedersen"**, dette ville skabe et problem, hvis man satte det direkte ind uden PreparedStatement, så ville man få en **baseQuery**, som blev stoppet for tidligt. Der ville prompte opstå en fejl, og programmet ville stoppe.
Derefter bliver der foretaget handlingen **prepStatement.executeUpdate();**
Dette afsender query'en til databasen, og returnerer et enten et rækketal(rowCount) eller i dette tilfælde, da der er tale om en **INSERT INTO**, gives der ikke noget tilbage. Der kommer **Statement.GENERATED.KEYS** ind i billedet; det giver den genererede nøgle tilbage, primærnøglen - kaldet *id*. For at få fat i denne, kaldes der en metode: **generatedKeys.getInt(1);** hvor generatedKeys er resultatet. Altså fås der noget tilbage alligevel!.
Resterne på nær en enkelt linje er blot **Try/Catch/Finally**-blokke, som foretager Error-Handling, og lukker forbindelsen til databasen. Den sidste linje er et kaldl til en metode, hvori alle **SalesOrderLine** objekter i ArrayListen i SalesOrder tilføjes til databasen; I programmet vedhæftes disse to igennem en ArrayListe med objekter. Men i databasen kobles de med en relation med en foreign key reference! Denne metode ser vi nærmere på et det andet kode eksempel.

```
public void createSalesOrder(long date, boolean deliveryStatus, long deliveryDate, float totalPrice,
        ArrayList<SalesOrderLine> orderLineArray, int customer_id) {


    PreparedStatement prepStatement = null;
    String baseQuery = "INSERT INTO sales_order "
            + "(sodate, delivery_status, delivery_date, total_price, customer_id) "
            + "VALUES (?, ?, ?, ?, ?)";
    Connection con = null;
    try {
        con = DBConnection.getInstance().getDBcon();
        prepStatement = con.prepareStatement(baseQuery, Statement.RETURN_GENERATED_KEYS);
        prepStatement.setLong(1, date);
        prepStatement.setBoolean(2, deliveryStatus);
        prepStatement.setLong(3, deliveryDate);
        prepStatement.setFloat(4, totalPrice);
        prepStatement.setInt(5, customer_id);

        @SuppressWarnings("unused")
        int rowsAffected = prepStatement.executeUpdate();
        try (ResultSet generatedKeys = prepStatement.getGeneratedKeys()){
            if(generatedKeys.next()){
                ID = generatedKeys.getInt(1);
                System.out.println(ID);
            }
```

```
43              } catch (Exception e) {
44                  System.out.println(e.getMessage());
45              }
46
47          } catch (Exception e) {
48              System.out.println(e.getMessage());
49          } finally {
50              if(con != null){
51                  try {
52                      DBConnection.closeConnection();
53                  } catch (Exception e2) {
54                      System.out.println(e2.getMessage());
55                  }
56              }
57          }
58          addOrderLinesToDB(orderLineArray, ID);
59      }
```

Herunder ses "**AddOrderLinesToDB**" metoden:

```
private void addOrderLinesToDB(ArrayList<SalesOrderLine> orderLineArray, int id) {
    for(SalesOrderLine sOL : orderLineArray){
        PreparedStatement prepStatement = null;
        String baseQuery = "INSERT INTO order_line (amount, product_id, sales_order_id) VALUES (?, ?, ?)";
        Connection con = null;
        try {
            con = DBConnection.getInstance().getDBcon();
            prepStatement = con.prepareStatement(baseQuery, Statement.RETURN_GENERATED_KEYS);
            prepStatement.setInt(1, sOL.getAmount());
            prepStatement.setInt(2, sOL.getProduct().getId());
            prepStatement.setInt(3, id);
            prepStatement.executeQuery();

        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            if(con != null){
                try {
                    DBConnection.closeConnection();
                } catch (Exception e2) {
                    System.out.println(e2.getMessage());
                }
            }
        }
```

Denne metode modtager en ArrayListe af SalesOrderLine, som kommer fra metoden ovenfor (se parametrene). Den tager også et id, således at man kan se hvilken salgsordre disse tilhører. Hertil falder logikken sammen fra de øvre lag; i GUI-laget indtastes der en række information om, hvilken kunde skal skrives på en given salgsordre, der tages en tid/dato fra computeren der udfører opgaven idet der trykkes "opret".
Der findes også en liste i GUI-laget, som indeholder rækken af ordrelinjer, som skal påføres salgsordren; og denne sendes med igennem controlleren til database-laget, til lagring. Altså er det fornuftigt, at disse to operationer opdeles i to metoder, som begge skal udføres, når der skal gemmes noget nyt i databasen; der skal gemmes i *sales_order* tabellen og i *order_line* tabellen.
På samme måde som før oprettes der forbindelse til databasen efter at have lavet en **baseQuery** med **INSERT INTO** igen. Notat: I dette tilfælde er der ikke brug for **GENERATED_KEYS**. Det eneste, der adskiller disse to metoder fundamentalt er, at denne gang er det hele omsvøbt af en ForEach, således at dette gøres for hele Array'et.

Herunder ses et udsnit af SalesOrderCTR, som styrer den centrale logik, primært mellem GUI laget og alle andre lag (DBAccess og og Model).

```
public SalesOrder createSalesOrder(long date, Boolean deliveryStatus, long deliveryDate,
        float totalPrice, ArrayList<SalesOrderLine> orderLineArray, int customer_id) {
    SalesOrder sO = new SalesOrder(date, deliveryStatus, deliveryDate, totalPrice, orderLineArray);
    sODB.createSalesOrder(date, deliveryStatus, deliveryDate, totalPrice, orderLineArray, customer_id);
    return sO;
}
```

Herunder findes en metode til at vise samtlige salgsordrer (og deres vedhæftede ordrelinjer). Til denne metode er der behov for at lave en **SELECT FROM WHERE** query, hvor vi

trækker al den nødvendige information ud af databasen. Dertil er det nødvendigt at lave en **JOIN**, idet at databaser ikke har andet end en reference imellem de tre typer af objekter; **SalesOrder**, **SalesOrderLine** og **Customer**. Ordre linjerne bliver ikke trukket ud af databasen her. Principielt det samme; Opret **PreparedStatement**, opret forbindelse. Denne gang udfører vi dog vores query først, og derefter anvender vi **ResultSet** til at trække informationen ud, og ligge dem i passende lokale variable.
Til sidst anvendes variablerne til at oprette objektet, hvorefter de tilføjes til et Array, som returneres til sidst i metoden.
Alt dette er i en **while(..)** blok, da skal tages flere objekter ud; der er tale om alle salgsordrer. Arrayet med alle salgsordrerne returneres til controlleren, som kalder til en anden, seperat metode, der henter alle ordrelinjerne ud på lignende vis. Alt returneres til GUI-laget til visning.

Det er database-siden der sørger for, at salgsordrerne får et id. Den tildeler dem et id, og vi påfører først dette id direkte i klassen, når vi henter dem ud af databasen igen. Det er denne, der bruges til at vedhæfte ordrelinjerne. Vi vælger *ikke* selv dette id. Ligeledes med customer id.
Det kan også bemærkes, at tider er beskrevet i datatypen Long; og er blot tiden i millisekunder fra The Epoch; 1. januar, 1970 kl. 00.00.00. Leveringstiden er samme tid som oprettelsestiden, blot ca. en uge længere frem.

```
public ArrayList<SalesOrder> showAllSalesOrders() {
    long date;
    Boolean deliveryStatus;
    long deliveryDate;
    float totalPrice;
    String orderList;
    ArrayList<SalesOrderLine> salesOrderLineList;
    Customer customer;
    int cid;
    String name;
    String address;
    String zipcode;
    String customerType;
    String city;
    String email;
    String phoneNumber;
    int sid;
    PreparedStatement statement = null;
    System.out.println("not working");
    String query = "SELECT sodate, delivery_status, delivery_date, total_price, order_list, customer.id AS
    ArrayList<SalesOrder> salesArray = new ArrayList<>();
    Connection con = null;
```

```
try {
    con = DBConnection.getInstance().getDBcon();
    statement = con.prepareStatement(query);
    ResultSet result = statement.executeQuery();

    while (result.next()) {
        date = result.getLong("sodate");
        deliveryStatus = result.getBoolean("delivery_status");
        deliveryDate = result.getLong("delivery_date");
        totalPrice = result.getFloat("total_price");
        orderList = result.getString("order_list");
        cid = result.getInt("customerid");
        name = result.getString("name");
        address = result.getString("saddress");
        zipcode = result.getString("zipcode");
        customerType = result.getString("ptype");
        city = result.getString("city");
        email = result.getString("email");
        phoneNumber = result.getString("phone_number");
        sid = result.getInt("salesorderid");
        System.out.println(name);

        customer = new Customer(cid, name, address, zipcode, customerType, city, email, phoneNumber);
        SalesOrder salesOrder = new SalesOrder(sid, date, deliveryStatus, deliveryDate, totalPrice, get
```

```
            salesArray.add(salesOrder);


        }

    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    return salesArray;
}
```

Fremtidigt Arbejde

Der er på nuværende tidspunkt lavet mange “lappe-løsninger”; Det er fx muligt at skrive totalprisen på en given salgsordre ind. Dette burde blive udregnet ud fra ordre-linjerne som bliver tilføjet. Mange af disse ting er ikke vurderet som essentielt for den centrale logik, og er derfor efterladt som de er. Det er heller ikke muligt at fjerne eller opdatere salgsordrer, produkter eller kunder. Man kan heller ikke finde/vise udvalgte salgsordrer, kunder mv., men man kan dog se alle af disse.
Der er kun foretaget central logik for den primære use-case.