Software Development

Underviser:

Mads Rosendahl

Assistent:

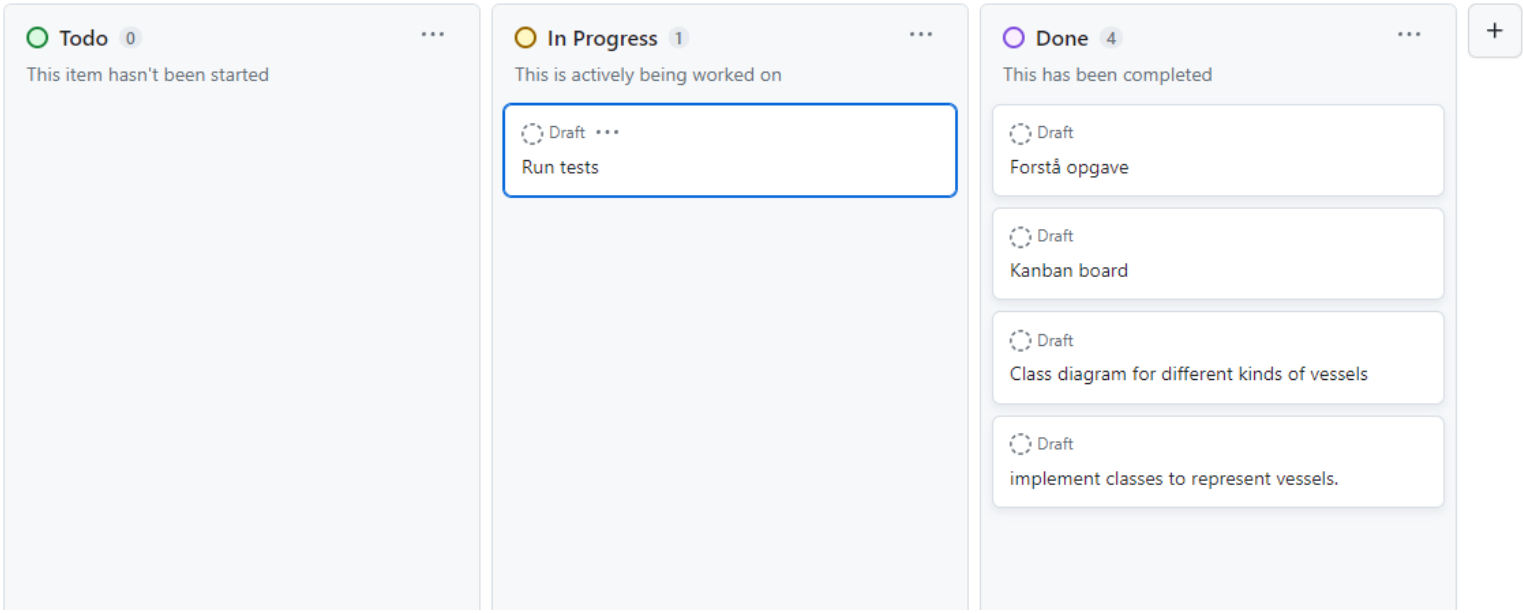
Nikolaj Robert Fuglsang Andersen

Gruppemedlemmer:

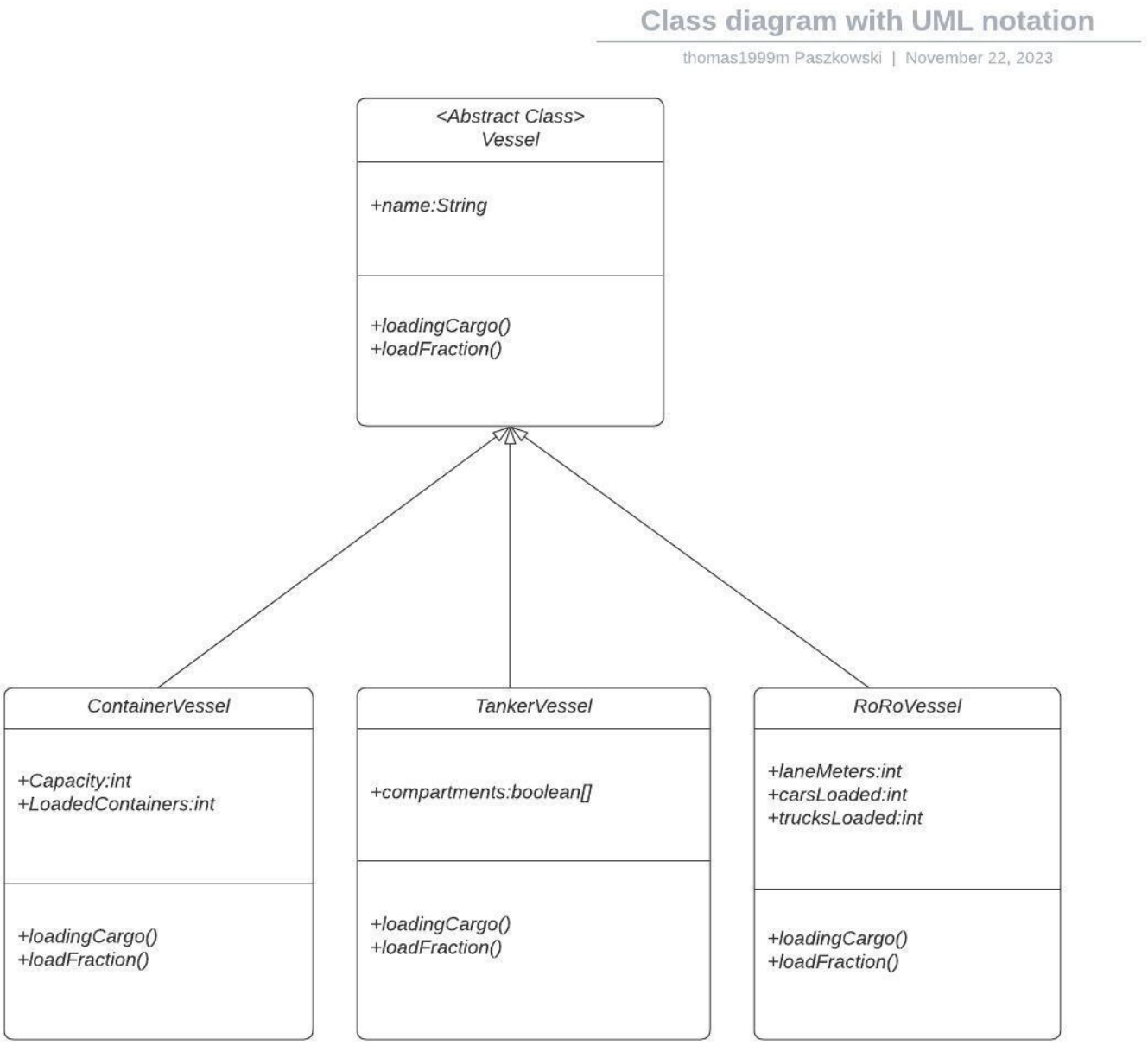
1. Thomas Axel Paszkowski Martyna, tapm@ruc.dk, studie nr. 74598
2. Tobias Brunton Torp Østergaard, tbto@ruc.dk, studie nr. 74155
3. Jia Hao Liang, jial@ruc.dk, studie nr. 74204

Portfolio 1

Part 1 - Kanban Board.



Part 2 - Make a class Diagram for the vessels.



Part 3 - Implement classes to represent these vessels.

(se kode bilag)

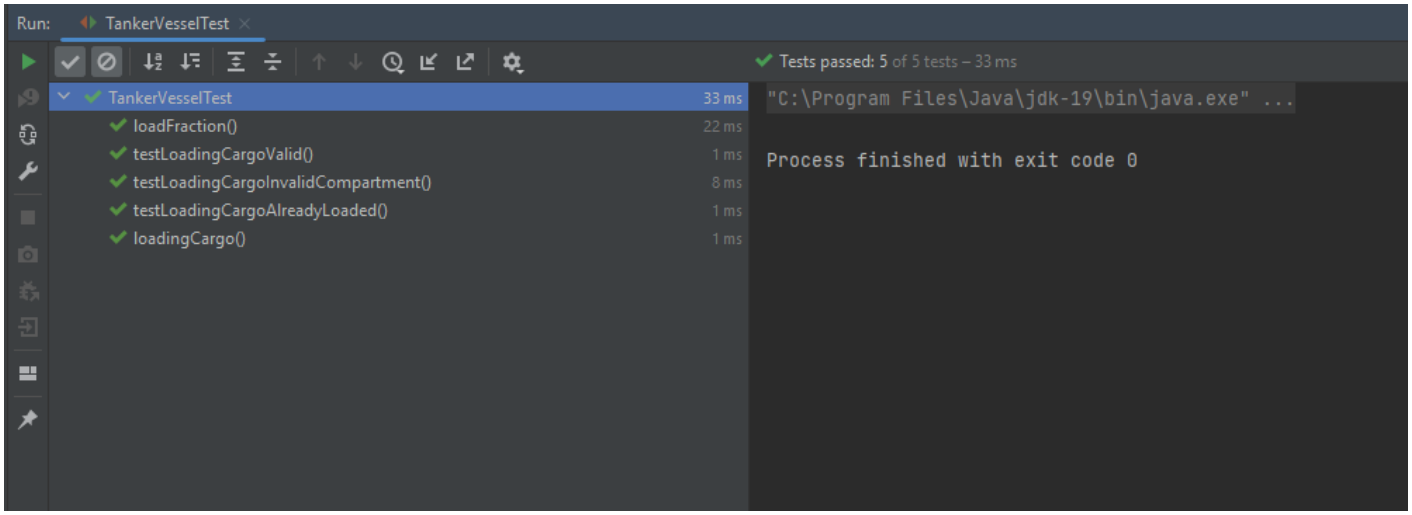
Part 4 - Provide unit tests for valid and invalid usage of the Tanker class.

Kode i bilag
I vores test af tanker class bliver der testet både for gyldig lastning, ugyldigt compartment, og dobbeltbelastning.

Under gyldig lastning ‘testLeadingCargoValid’ er formålet at se om 'loadingCargo' metoden bliver korrekt håndteret med en gyldig lastning anmodning. Vores forventede resultat var at metoden blev fuldført uden problemer, hvilket også var resultatet her.

Under ugyldigt compartment ‘testLoadingCargoInvalidCompartment’ er formålet at se om 'loadingCargo' vil tage imod at blive fyldt op i en compartment der ikke findes. Vores forventede resultat var, at der ville opstå et problem, hvilket også var resultatet her. Vi brugte assertThrows for at testen vidste der skulle opstå et problem her.
Under ugyldigt compartment 'testLoadingCargoAlreadyLoaded' er formålet at se om 'loadingCargo' vil tage imod at blive fyldt op i samme compartment 2 gange. Vores forventede resultat var, at der ville opstå et problem, hvilket også var resultatet her. Vi brugte assertThrows for at testen vidste der skulle opstå et problem her.

Herunder kan vi se vores færdige unittest, som fortæller os alt passer.



For at løse opgaven har vi fra undervisningen benyttet (Kursusgang refereret med nr.):

2. OOP, inheritance, polymorphism, generics

- https://www.youtube.com/watch?v=Ft88V_rDO4I
- https://youtu.be/0xw06loTm1k
- Exercise with animal class.

3. Exceptions, software testing and JUnit, Git

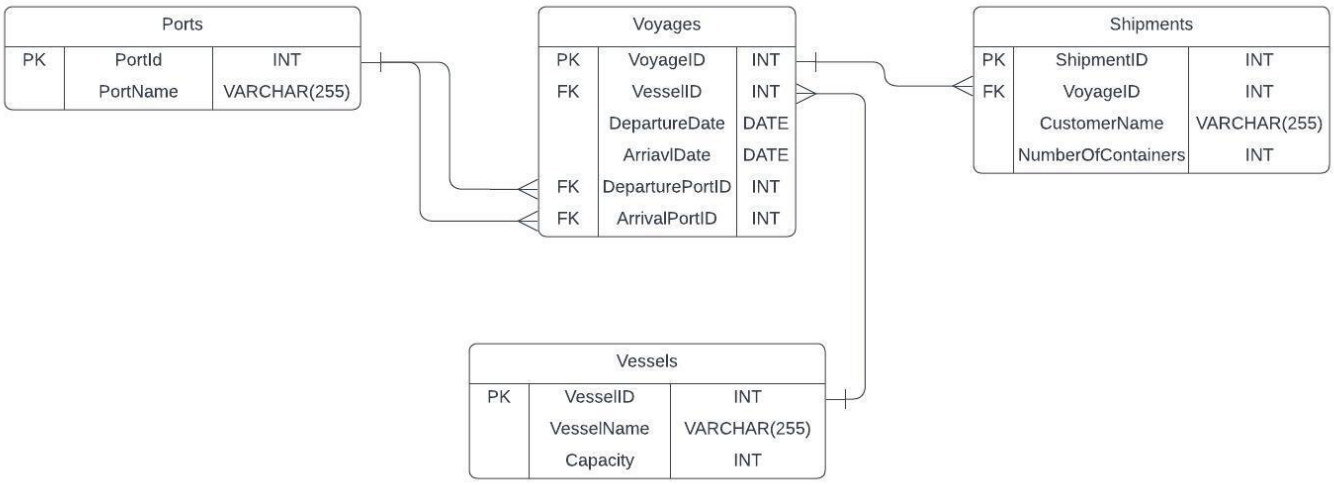
- Exercise with animal class.

4 Kanban, Use-case, UML, Generics, List

Slide 5 - kanban board

Portfolio 2

Part 1 - E/R Diagram



Part 2 - Create Table

Create the tables:

```
public void createTables() { //metode til at oprette database tabeller.
   cmd("CREATE TABLE IF NOT EXISTS Ports (PortId INT PRIMARY KEY, PortName VARCHAR(255) NOT NULL);");//
   cmd("CREATE TABLE IF NOT EXISTS Vessels (VesselID INT PRIMARY KEY, VesselName VARCHAR(255) NOT NULL, Capacity INT NOT NULL);");
   cmd("CREATE TABLE IF NOT EXISTS Voyages (VoyageID INT PRIMARY KEY, VesselID INT, DepartureDate DATE, ArrivalDate DATE, VesselName VARCHAR(255) NOT NULL, DeparturePortID INT, ArrivalPortID INT);");
   cmd("CREATE TABLE IF NOT EXISTS Shipments (ShipmentID INT PRIMARY KEY, VoyageID INT, CustomerName VARCHAR(255) NOT NULL, NumberOfContainers INT NOT NULL);");
```

Insert data to the tables:

Se kode

Part 3 and part 4 - Create a query to verify that no voyages have more shipment cargo than the capacity.

SELECT V.VoyageID, SUM(S.NumberOfContainers) AS TotalContainers, VE.Capacity

Vælger voyage-ID'et, beregner den samlede sum af containere for den voyage og vælger kapaciteten af det skib, der er tildelt for voyagen.

FROM Shipments S

Starter med 'Shipments tabellen, for at få data om forsendelsen.'

JOIN Voyages V ON S.VoyageID = V.VoyageID

Joiner 'voyages' tabellen for at matche forsendelser med de respektive voyages.

JOIN Vessels VE ON V.VesselID = VE.VesselID

Joiner 'Vessels' tabellen for at få information om skibets kapacitet, der er blevet tildelt til hver voyage.

GROUP BY V.VoyageID

Gruperer resultaterne efter 'VoyageID', så vi kan bruge SUM funktionen

HAVING SUM(S.NumberOfContainers) > VE.Capacity;

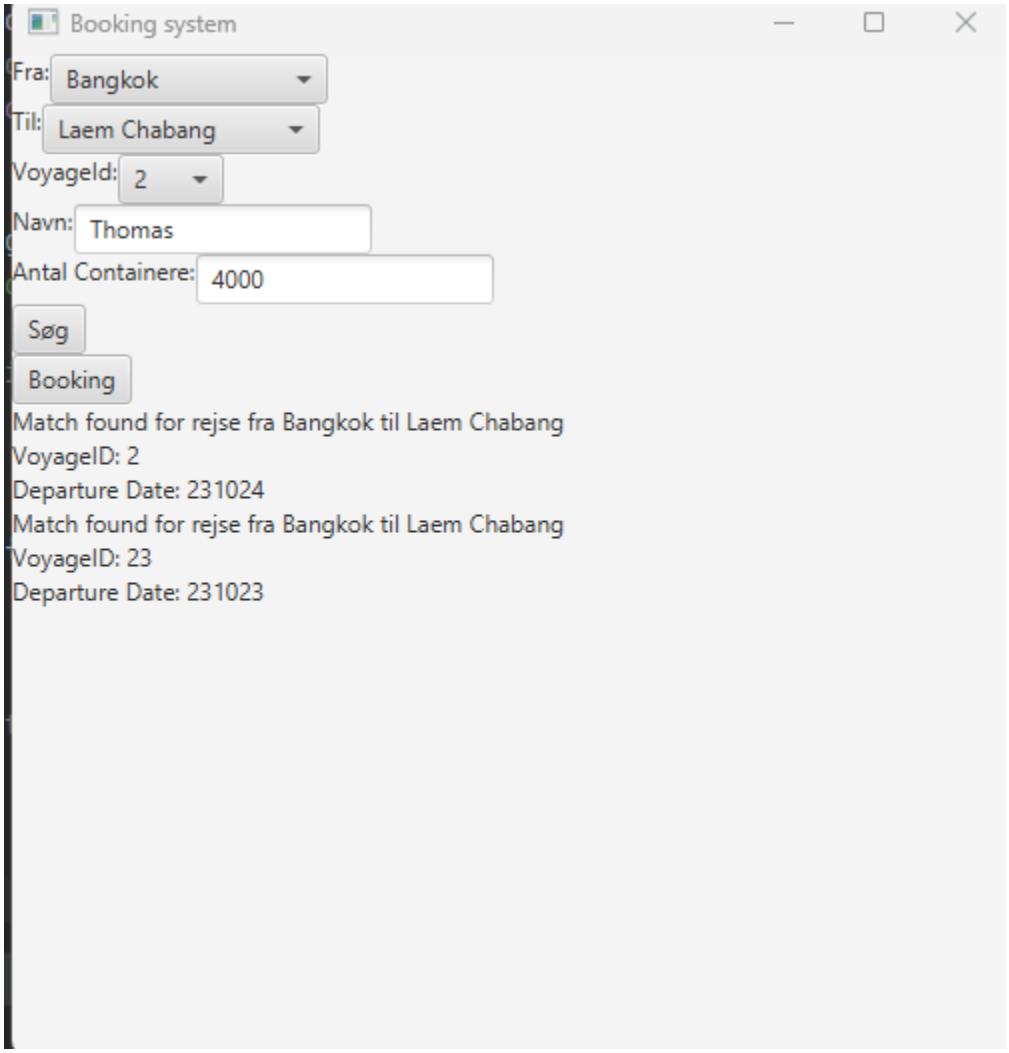
Filtrerer den grupperede resultater for kun at inkludere dem, hvor den samlede sum af bookede containere overstiger skibets kapacitet.

Part 5 - User interface in JavaFX

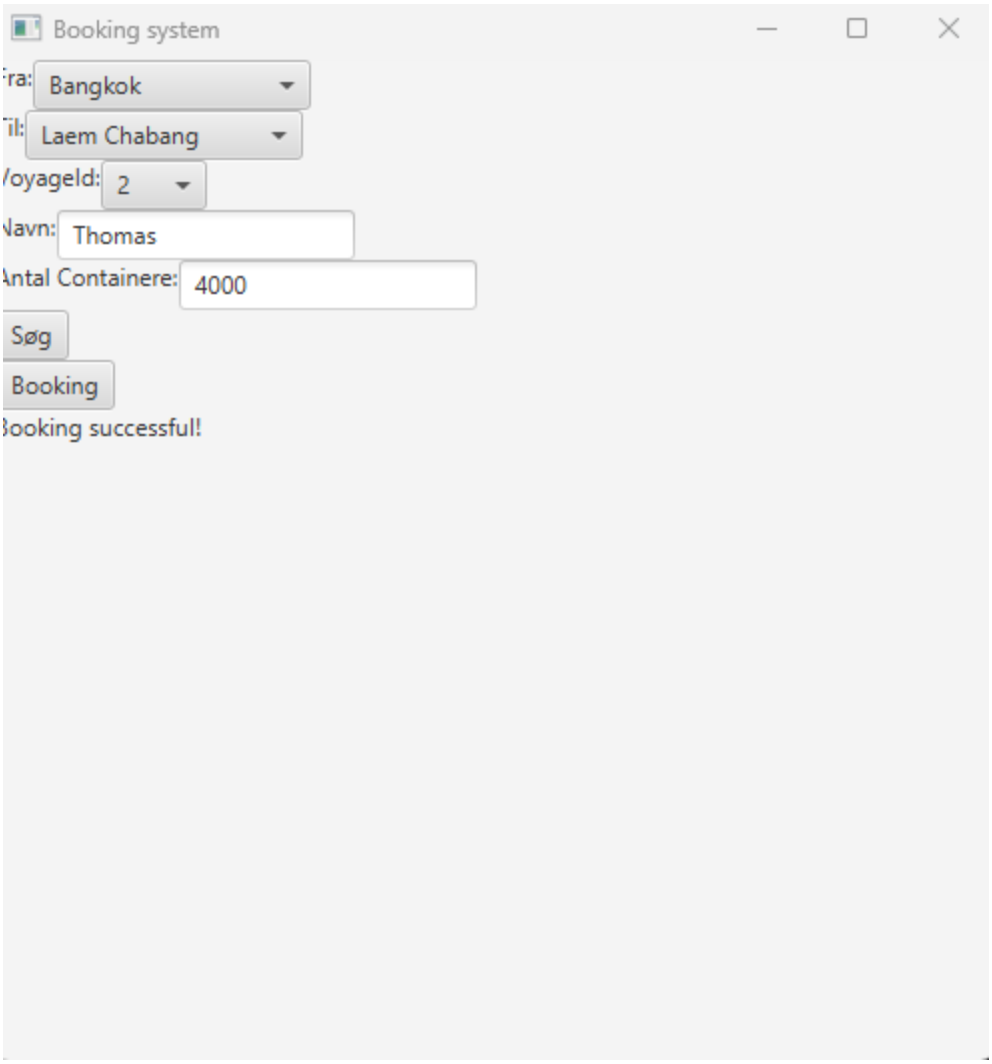
Se kode.

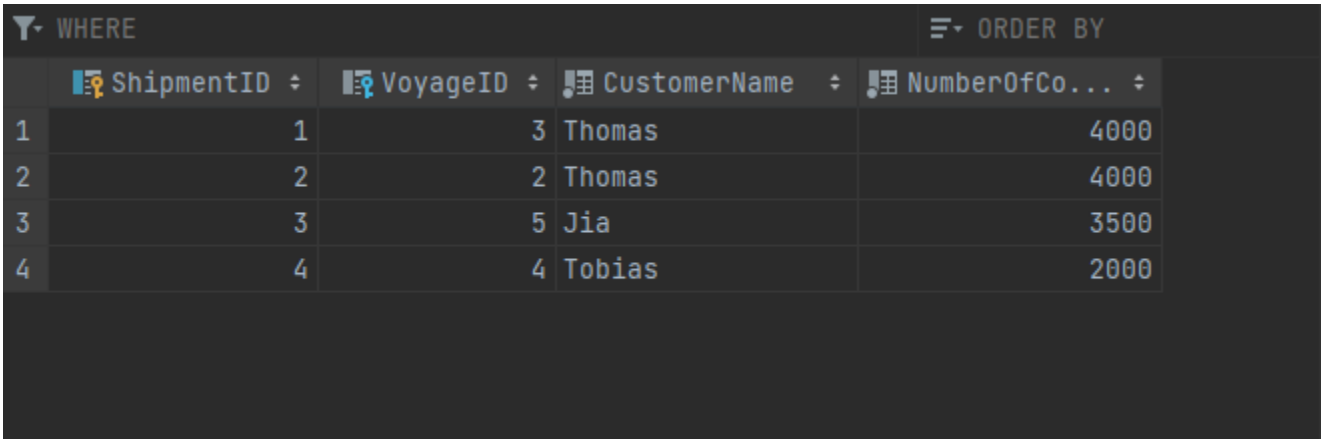
Part 6 - Document & Screenshots

Document runs of the program with screenshots. Document that you can add shipments to the database using your system.

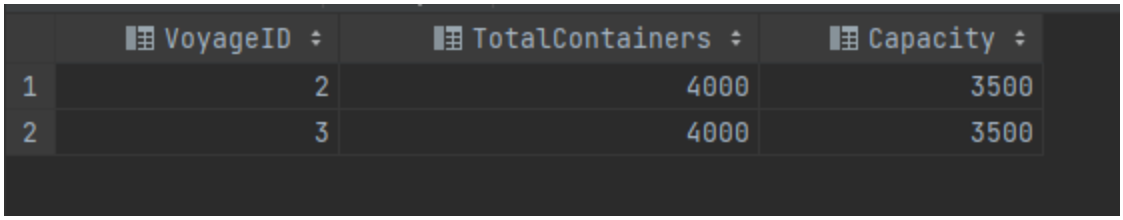


Der indsættes 4000 containere, så der kommer til at være for mange af dem. Dermed kan vi nu teste SQL erklæringen (statement), for tjekke af overbookede afgange.





Nu hvor der er blevet indsat noget data inde i databasen, kan vi tjekke om SQL erklæring (statement) virker som den skal.



	VoyageID	TotalContainers	Capacity
1	2	4000	3500
2	3	4000	3500

På udsnittet kan vi se at vi har to skibe, hvor mængden af containere overskrider kapaciteten af skibene, ved hjælp af vores query.

For at løse opgaven har vi fra undervisningen benyttet (Kursusgang refereret med nr.):

10. Database retrieval and manipulations

- **Slide s.13,15,18,19,20,22**

13. javaFX programming, MVC

- Example from lecture
- MVC example from lecture

11. Java Database Connectivity and javaFX

- Read csv file to produce SQL statements.

12. JavaFX programming, lambda expressions

- Solution to exercise GUI
- Java example from lecture

Portfolio 3

Part 1 - Read the file of the current route network and represent it as a graph

For at læse rutenetværket og repræsentere det som en graf, bruger vi klasserne:
Vertex{} & Edge{} til at repræsentere grafens elementer, vertex repræsenterer et knudepunkt, (en havn) og edge klassen repræsenterer en rute mellem to havne med en bestemt vægt (rejsetiden)

"class Graph {

HashMap<String, Vertex> vertices; // Opbevaring af knudepunkter.

HashSet<Edge> edges; // Opbevaring af kanter."

graph indeholder og håndterer alle knudepunkter og kanter.

"void insertEdge(String from, String to, int weight) // Tilføjer kanter til grafen."

'insertEdge'-metoden bruges til at tilføje en ny rute til grafen.

}

For at indlæse samt opbygge grafen benytter vi:

'static void readAndStoreGraph(Graph graph, String fileName) throws IOException'

Her håndterer vi indlæsningen af rute data fra en fil og anvender 'insertEdge' metoden til at opbygge grafen.

Part 2 - Check that the graph is connected so that there is a possible route between any two ports.

For at sikre, at grafen er sammenhængende, som betyder at der findes en rute mellem ethvert par af havne, bruger vi følgende kode:
'boolean isConnected()'
'void visitDepthFirst(Vertex v, Set<Vertex> visited)'
'isConnected' burger dybde først søgning (implementeret i 'visitDepthFirst') til at besøge alle knudepunkter. Hvis vi kan nå alle knudepunkter fra et vilkårligt startpunkt, er grafen sammenhængende.

Part 3 - Construct the minimum spanning tree of the graph.

For at konstruere minimum spanning tree (MST) af grafen, benytter vi:
'Set<Edge> minimumSpanningTree()'
'minimumSpanningTree' metoden anvender en algoritme til at finde den letteste kant, der kan tilføjes til MST, og gentager denne proces, indtil alle knudepunkter er inkluderet i træet.

Forklaring af Prims algoritme:

1. Algoritmen starter med at tilføje et tilfældigt knudepunkt til 'frontier'-sættet, for at repræsentere de knudepunkter, der er inkluderet i MST indtil videre.

2. i hvert trin a while-løkken, gennemgår algoritmen alle kanter og vælger den letteste kant, der forbinder et knudepunkt i 'frontier' med et knudepunkt udenfor 'frontier'.

3. Den valgte kant og dens tilhørende slutvertex tilføjes til MST og 'frontier'.

4. Processen gentages, indtil alle knudepunkter er blevet tilføjet til MST.

Hvad er kompleksiteten af algoritmen?

Vi har to løkker, hvor den yderste løkke fortsætter indtil alle knudepunkter er inkluderet i MST, eller der ikke er flere kanter at tilføje, kompleksiteten er O(V), hvor V er antallet af knudepunkter. Den indre løkke har en for-løkke, der itererer over alle kanter for at finde kanten med den mindste vægt, som forbinder 'frontier' med resten af grafen. Denne løkke gennemgår alle kanter, så dens kompleksitet er O(E). den samlede kompleksitet af algoritmen er O(VE), da vi for hvert knudepunkt(V) scanner alle kanter (E).

For at løse opgaven har vi fra undervisningen benyttet (Kursusgang refereret med nr.):

14. Threads, streams, graphics

Her har vi taget inspiration fra løsningen fra slide 4. Til at indlæse data fra en tekstfil

16. Graph representation, traversal

her har vi taget udgangspunkt i 'Program code from lecture'

17. Dijkstra, minimum spanning tree

Her har vi i slides(slide 31) fundet løsningen til at konstruere vores 'minimum Spanning Tree.'