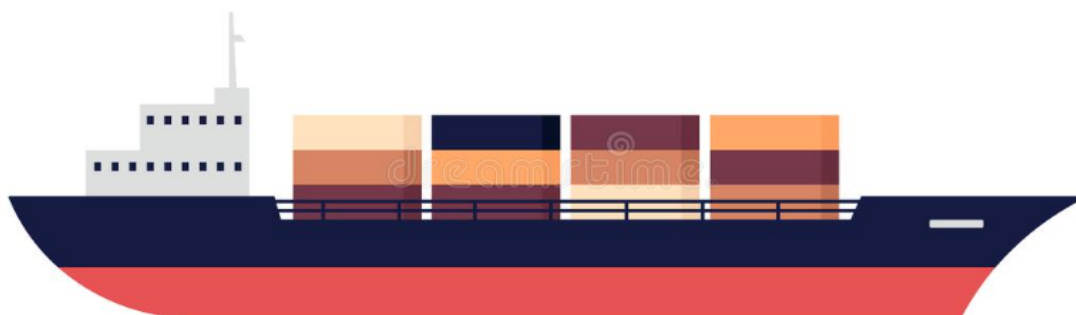


# Subject Module Course 2: Software Development



## Link til Github:

Portefølje 1: <https://github.com/LauraLSJN/Portofolie1EKSAMEN.git>

Portefølje 2: [https://github.com/CWulffeld/Portfolio2EKSAMEN\\_v1.git](https://github.com/CWulffeld/Portfolio2EKSAMEN_v1.git)

Portefølje 3: <https://github.com/LauraLSJN/Portofolie3SDVersion2.git>

Navn	Studienummer
Anna Borg aborg@ruc.dk	71703
Christine Van Wulffeld cvanw@ruc.dk	71998
Laura Sofie Juel Nielsen lsjn@ruc.dk	72005

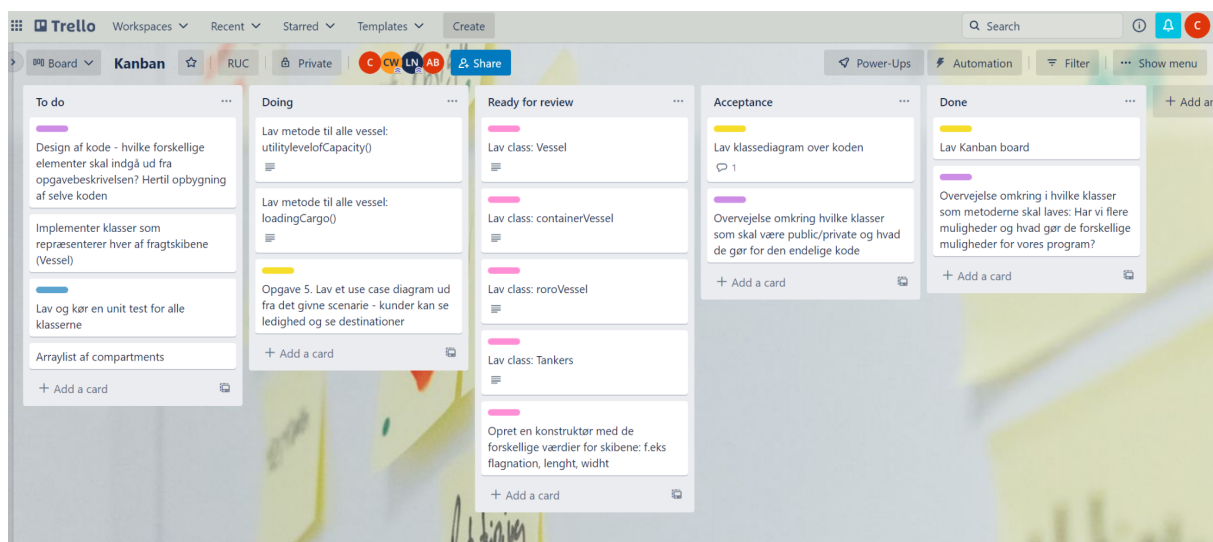
# Portefølje 1

Portefølje 1 var oprindeligt udarbejdet af følgende studerende; Anna Borg, Christine Van Wulffeld, Laura Sofie Juel Nielsen og Caroline Hornesby Vargas. Efterfølgende har Anna Borg, Christine Van Wulffeld og Laura Sofie Juel Nielsen rettet porteføljen samt koden til.

## Part 1: Kanban Board

Vi har udarbejdet et kanban board, hvor vi har haft mulighed for at følge vores “product flow”. Kanban Board er et redskab til at skabe overblik og forbedre planlægning af projektet. Vi har valgt at anvende websiden Trello til at udarbejde vores Kanban Board (Anderson & Carmichael, 2016).

Kanban boarded er udarbejdet efter processen i at designe, implementere og teste et system. Denne proces har vi valgt at inddele i følgende trin: 1) To do - indeholder alle opgaver som skal udføres. 2) Doing - repræsenterer at en opgave er under udarbejdelse. 3) Ready for review - Opgaven er udarbejdet og er klar til review. 4) Acceptance - Opgaven skal gennemgås. I Ready For Review kan der forekomme rettelser, hvis dette ikke er tilfældet placeres opgaven i den femte og sidste kategori 5) Done.



Ydermere er det muligt at tilføje personer til opgaverne, hvilket repræsenterer den ansvarlige for den specifikke opgave.

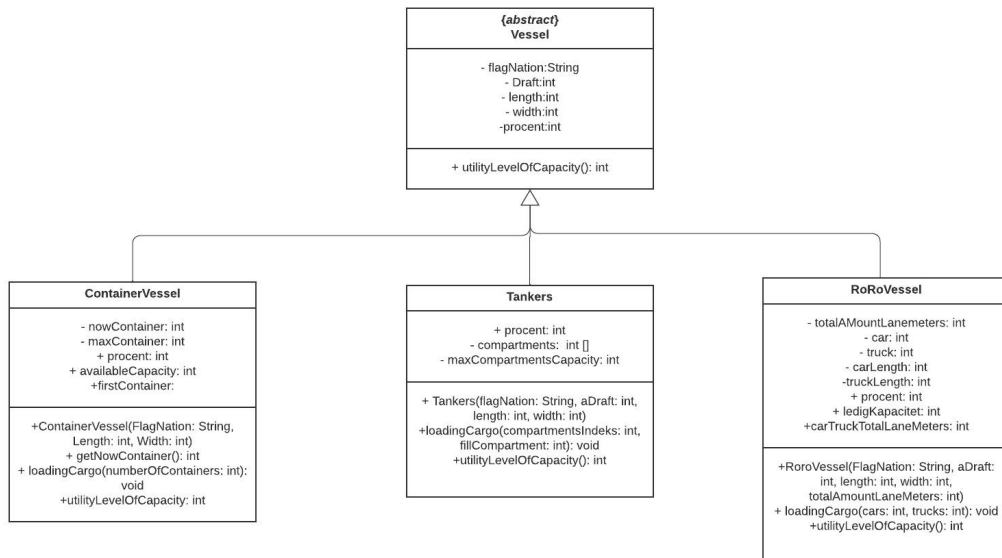
Projektgruppens første opgave var at udarbejde et klassediagram. Klassediagrammet blev udarbejdet i en tidlig projektfase, hvilket gav projektgruppen mulighed for at definere

variabler og metoder, som skulle indgå i programmet. Efter klassediagrammet var reviewet, begyndte projektgruppen at implementere klasserne og metoderne, som blev identificeret i klassediagrammet.

Sideløbende med kodens udvikling vil der blive gjort brug af JUnit testing. Her vil vi kunne isolere dele af vores kode for at se om de enkelte dele fungerer efter hensigten. JUnit testing implementeres samtidigt med vi udvikler programmet for at sikre at koden virker optimalt og for at finde potentielle fejl tidligt i processen.

## **Part 2: Klassediagram**

I nedenstående klassediagram fremgår 4 klasser, hvoraf relationen generalization anvendes. Generalization beskrives som en “is-a” relation, hvor der fremgår en relation mellem en sub- og en super-klasse. Begreberne child- og parenting klasse anvendes ligeledes. En sub-klasse kan tilgå og anvende super-klassens attributter (pånær private), metoder og endvidere override super-klassens metoder. Ved overriding kan en sub-klasse give en specifik implementering af en metode, som allerede er givet ved super-klassen. En sub-klasse kan dermed tilgå og anvende ovenstående, hvilket betegnes som nedarvning i Java (Goodrich et al., 2014). Generalization-relationen illustreres ved pilene med et hvidt pilehoved, som starter fra en sub-klasse og afslutningsvis peger på en super-klasse. Dette betyder at Vessel er en superklasse, hvoraf containerVessel, RoroVessel og Tankers er sub-klasser, som nedarver fra superklassen Vessel. Derudover fremgår det, at Vessel er en abstract klasse, og derved står klassens navn i kursiv. Vessel-klassen indeholder én abstract metode samt fem variabler (Seidl et al., 2015).



## Part 3: loadingCargo() & utilityLevelOfCapacity()

Der er foretaget ændringer på alle klassernes loadingCargo() metoder, i henhold til aflevering af portefølje 1.

I opgavebeskrivelsen fremgår det, at projektgruppen skal anvende nedarvning og polymorphism. Dette anvendes ved at superklassen Vessel, har 3 sub-klasser: RoroVessel, containerVessel og Tankers. Vessel er en abstrakt klasse med en abstrakt metode; utilityLevelOfCapacity(). Da Vessel indeholder en abstrakt metode, kan der ikke skrives logik i metodens body, samt metoden skal overrides i de tre subklasser. De tre sub-klasser "Overrider" super-klassen metode: utilityLevelOfCapacity. Derudover indeholder de tre sub-klasser metoden: loadingCargo(). Metoden loadingCargo() er void og har ikke en returnværdi modsat returnere metoden utilityLevelOfCapacity() en int.

### 3.1 ContainerVessel:

```

public void loadingCargo(int numberOfContainers) {
    availableCapacity = maxContainer - nowContainer;
    if (nowContainer < maxContainer) {
        if (numberOfContainers <= availableCapacity) {
            nowContainer += numberOfContainers;
        } else {
            System.out.println("Antallet af containere ikke muligt. Max. kapacitet: 10, Min. kapacitet: 0");
        }
    } else {
        System.out.println("Der er allerede fyldt op");
    }
}

if (nowContainer == maxContainer) {
    System.out.println("Der er præcis 10");
}
}

```

Metoden `loadingCargo()` har følgende parameter; `int numberOfContainers`, hvor `numberOfContainers` angiver hvor mange containers der skal tilføjes til `containerVessel`. Først tildeles den globale variabel "`availableCapacity`" en værdi, som er differencen mellem variablerne; `maxContainer` og `nowContainer`. Derefter forekommer der et "nested" if-statement, som er en eller flere if-statement, indlejret i en anden if-statement.

Det første if-statement tjekker om `nowContainer` er mindre end `maxContainer`, hvis dette er tilfældet, tjekker den anden if-statement om `numberOfContainers` er mindre eller lig med `availableCapacity`. Hvis disse er sande, øges `nowContainers` med `numberOfContainers`.

Efter det "nested" if-statement, forekommer et nyt if-statement, som tjekker om `nowContainer` og `maxContainer` har samme værdi. Hvis dette er tilfældet, vil en meddelelse blive printet i konsollen.

```
@Override
public int utilityLevelOfCapacity() {
    procent = nowContainer * 100 / maxContainer;
    System.out.println("Andelen af ContainerVessel der er fyldt: " + procent + "%");
    return procent;
}
```

Metoden `UtilityLevelOfCapacity()` returnerer en `int`. Denne metode udregner andelen af fyldte containers, dette er udregnet ud fra procent regneregler.

## Tankers:

```
public void loadingCargo(int compartmentsIndeks, int fillCompartment) {
    if (compartmentsIndeks > 9) {
        System.out.println("Tankeren har 1-10 compartments");
    }else if (compartmentsIndeks < 0){
        System.out.println("The tankers have 1-10 compartments");
    }
}

if (compartments[compartmentsIndeks] == 0) { //Fylder kun på, hvis værdien ved indeks x er 0
    //Plusser ikke op i antal, der allerede er lagt i. Vi kan derved ikke fylde 2+4 i samme indeks (compartment)
    if (fillCompartment <= maxCompartmentCapacity){
        compartments[compartmentsIndeks] += fillCompartment;
        System.out.print(" Compartments: ");
        for (int com : compartments) {
            System.out.print(com);
        }
    } else {
        System.out.println("Du overskrider max level for compartment");
    }
    System.out.println();
}else{
    System.out.println("Der er allerede en på pladsen");
}
}
```

Metoden `loadingCargo()` tager to parametre; `int compartmentsIndeks` og `int fillCompartments`. Metoden starter med at en if-else statement, som tjekker om `compartmentsIndeks` er større end 9 eller mindre end 0. Dette er implementeret, da arrayet `compartments` har længde på 10. Derefter forekommer et “nested” if-statement, hvor det første if-statement tjekker om værdien på det givne indeks (`compartmentsIndeks`) i `compartments` array er lig 0. Hvis dette er tilfældet tjekker det næste if-statement om `fillCompartments` er mindre eller lig `maxCompartmentsCapacity`. Er dette ligeledes tilfældet indsættes det `fillCompartments` værdien i det givne `compartmentsIndeks`. Derefter printes arrayet, i et for-each loop.

```
@Override
public int utilityLevelOfCapacity() {
    int count = 0;
    procent = 0;
    for (int com : compartments) {
        if (com == 0) {
            count++;
            procent = count * 100 / compartments.length;
        }
    }
    System.out.println("Andel af tomme compartments: " + procent + "%");
    return procent;
}
```

Metoden `UtilityLevelOfCapacity()` returnerer en `int`. Denne metode udregner andelen af tomme compartments, ved at løber arrayet igennem (for-each loop), hvor variabelen `count` øges hver gang en compartment i arrayet er 0. `Count` anvendes til at udregne andelen af tomme compartments.

### **RoroVessel:**

```

public void loadingCargo(int cars, int trucks) {
    int newLoadCar = cars;
    int newLoadTruck = trucks;
    int newLoadLaneMeters = (newLoadCar * carLength) + (newLoadTruck * truckLength); //TotalLaneMeters der ønskes at tilføjes
    carTruckTotalLaneMeters = (car * carLength) + (truck * truckLength); //TotalLaneMeters for alle car og truck tilføjet
    ledigKapacitet = totalAmountLaneMeters - carTruckTotalLaneMeters; //LedigKapacitet tilbage

    if (ledigKapacitet >= 0 && totalAmountLaneMeters > carTruckTotalLaneMeters) {
        if (newLoadLaneMeters < ledigKapacitet) { //Den ønskede værdi der skal tilføjes skal være mindre end ledig kapacitet
            car += newLoadCar; //tilføjer newLoadCar til variabelen car
            truck += newLoadTruck; //tilføjer newLoadTruck til variabelen truck
            System.out.println("Car: " + car + " Truck: " + truck);
            System.out.println("newLoadLaneMeters: " + newLoadLaneMeters);
            System.out.println("carTruckTotalLaneMeters: " + carTruckTotalLaneMeters);
            carTruckTotalLaneMeters = (car * carLength) + (truck * truckLength); //Opdaterer carTruckTotalLaneMeters
            ledigKapacitet = totalAmountLaneMeters - carTruckTotalLaneMeters; //Opdaterer ledigKapacitet
            System.out.println("LedigKapacitet: " + ledigKapacitet);

            System.out.println();
        } else {
            System.out.println("Der er ikke plads");
        }
    }
}
}

```

Metoden `loadingCargo()` har to parameter; `int cars` og `int trucks`. Værdierne for disse to parametre tildeles i følgende lokale variabler; `int newLoadCar` og `int newLoadTruck`. Dette muliggør at adskillelse af hvor mange cars og trucks der er i forvejen, og hvor mange der tilføjes. Derefter udregnes den samlede længde af “lane meters” for det samlede antal af cars og truck og det samlede antal “lane meters” for cars og trucks angivet i metodens parameter. Disse udregner anvendes til at finde den ledige kapacitet på RoroVessel. Derefter forekommer der et nested if-statement, hvor den første if-statement tjekker om den ledige kapacitet er større eller lig med 0, og om RoroVessels maks “lane meters” er større end alle tilføjet cars og trucks totale “lane meters”. Hvis dette er tilfældet, tjekker det andet if-statement om det metodens parameters samlet “lanemeters” er mindre end den ledige kapacitet på RoroVessel. Hvis dette ligeledes er tilfældet, øges det eksisterende antal cars og trucks med det antal cars og trucks angivet i parameteren. Derefter opdateres den samlede mængde af lane meters for cars og trucks samt den ledige kapacitet på RoroVessel.

```

@Override
public int utilityLevelOfCapacity() {
    procent = carTruckTotalLaneMeters * 100 / totalAmountLaneMeters; //Procent andel som anvendes
    System.out.println("Andel af lanemeters fyldt: " + procent + "%");
    return procent;
}

```

Metoden `utilityLevelOfCapacity()` returnere en `int`. For at udregne andelen af optaget lanemeters anvendes variablerne; `carTruckTotalLaneMeters` og `totalAmountLaneMeters`.

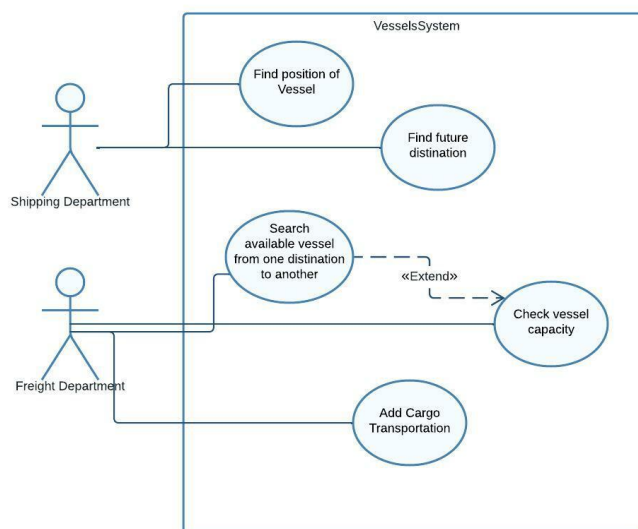
## Part 4: JUnit Test

Vi anvender JUnit testing til at teste om metoderne og klasserne udfører det vi forventer. JUnit testing er udarbejdet i klassen “MainTest”. Det forventelige resultat sammenlignes med det faktiske, til dette anvendes assertEquals(). Vores teststrategi er at teste for 0, 1 og flere kald af loadingCargo metoden, hvortil vi også tester om metoden utilityLevelOfCapacity returnere 0, 50 og/eller 100%.

JUnit testen er et godt instrument, til at teste om de enkelte dele af programmet faktisk virker. JUnit testen er en god måde hvorpå at vi kan argumentere for at vores kode har en høj pålidelighed. Vi er i forlængelse af dette opmærksomme på, at vi ikke kan benytte JUnit testen til at skabe komplet sikkerhed omkring at programmet virker.

## Part 5: Fremtidig brug

Use case diagrammet viser hvilke funktionalitet aktørerne skal kunne anvende ved udvidelse af systemet. Der fremgår to primær aktører; Shipping Department og Freight Department. Aktøren Shipping Department har en association til to use cases; Find position of vessel og Find future destination. Aktøren Freight Department har en association til use casene: Search available vessel from one destination to another, Check vessel capacity og Add cargo transportation. Der forekommer en “extend” association mellem use casene; search available vessel from one destination to another og Check vessel capacity. Da det skal være muligt for use casen Check vessel at anvendes selvstændigt og i forlængelse af base use casen (Seidl et al., 2015).





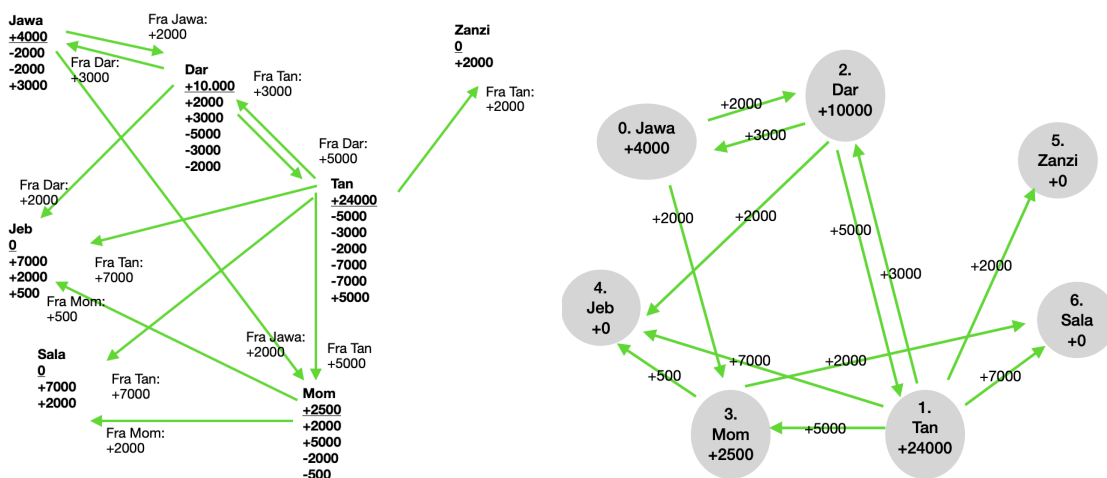
Ydermere er der implementeret “dummy” metoder i programmet, som repræsenterer de 5 forskellige use-case, hvilket kan implementeres på sigt.

## Portefølje 2

I denne portefølje er der implementeret kode givet af Mads Rosendahl. Denne kode er hentet fra Software Development 2, Moodle side. Koden der er implementeret er klasserne: AdjacencyGraph, Vertex og Edge.

### Part 1: Overskydende eller manglende containere ved hver havn

Hvad gør vi når vi skal have det korrekte antal containere tilbage igen? Det nemmeste ville være at reverse dem, men kan dette kan gøres på en smartere måde? Det kan gøres på en bedre måde, da man ved at reverse flowet risikere at flytte tomme containere, som har en omkostning på \$100 pr. tom container, jævnfør minimum cash flow problem angivet i opgavebeskrivelsen. I og med at firmaet ønsker at bruge mindst mulige penge på at flytte tomme containers, vil vi derfor finde en løsning, hvorpå vi kan flytte det mindst mulige antal af containere og samtidig ende med at have samme antal containere i hver havn, som da vi startede. Vi har derfor udarbejdet to pilediagrammer og tre tabeller som illustrerer flow-tabellen i opgavebeskrivelsen. Pilediagrammerne og tabellerne giver et overblik over antal flyttet container i henhold til start- og slutdestination. Disse pilediagrammer og tabeller viser også overskydende og manglende på tomme containere ved hver havn.



Start		Slut		Slut saldo	
Port	Total	Port	Total	Port	Total
Jawa	4000	Jawa	3000	Jawa	-1000
Tan	24000	Tan	5000	Tan	-19000
Dar	10000	Dar	5000	Dar	-5000
Mom	2500	Mom	7000	Mom	4500
Zan	0	Zan	2000	Zan	2000
Jeb	0	Jeb	9500	Jeb	9500
Sala	0	Sala	9000	Sala	9000
I alt	40500	I alt	40500		

Til at finde overskydende og mangel af containere har vi anvendt Directed Graph, som repræsenteres af en Adjacency List. I grafen forekommer der Vertex, Edges og Weight som repræsentere, havne, flow og antal containere pr. flow.

```

public void runVertexAndEdges() {
    for (Vertex v : adjDirectedG.Vertices) {
        for (Edge e : v.OutEdge) {
            Vertex f = e.from;
            Vertex t = e.to;
            int w = e.weight;

            int idxf = adjDirectedG.Vertices.indexOf(f);
            int idxt = adjDirectedG.Vertices.indexOf(t);
            saldo[idxf] -= w;
            saldo[idxt] += w;
            names[idxf] = v.toString();
        }
    }
    System.out.println();

    for (int i = 0; i < saldo.length; i++) {
        System.out.println("port: " + names[i] + " surplus: " + saldo[i]);
    }
    System.out.println();
}

```

```

port: Jawa surplus: -1000
port: Tan surplus: -19000
port: Dar surplus: -5000
port: Mom surplus: 4500
port: Jeb surplus: 9500
port: Zan surplus: 2000
port: Sal surplus: 9000

```

I klassen Flow har vi udarbejdet metoden runVertexAndEdges() for at finde overskydende og mangel på containers ved hver havn. Metoden løber igennem alle havne (Vertex) i arraylisten med deres tilhørende flows (Edges) ved brug af et nested for-each loop.

Metoden anvender et nested for-each loop, hvor den løber igennem alle havnene (Vertex) som ligger i én arraylist. Ved hver havn, løber metoden igennem det andet for-each loop som gennemløber en arraylist som indeholder flows (Edge). Ved hvert flow tildeles Vertex f værdien fra startdestinationen, Vertex t værdien af slutdestinationen og int weight tildeles antallet af containere der flyttes. Derefter anvendes Vertex f og Vertex t, til at ligge værdien for indeks f og t (fra arraylisten Vertices) i to nye lokale variabler (idxf og idxt).

Disse to nye variabler anvendes derefter til at finde et specifikt indeks (f og t) i saldo arrayet, hvortil weight minuses i saldo[idxf], da dette er startdestinationen og pluses i saldo[idxt], da

dette er slutdestinationen. Efter det nested for-each loop, printes saldo listen ud ved brug af et for loop.

## **Part 2: Pris for reverse flow**

Nedenstående har vi lavet en udregning af, hvad det koster hvis vi blot sender alle de tomme containere tilbage, til den havn som de oprindeligt blev fragtet fra.

40.500 containers \* \$100 = \$4.050.000

## **Part 3: Metode der returnere alle containers til oprindelige havne**

Vores program kan flytte containere med følgende minimum cost:

25.000 overskydende containers \* 100 = \$2.500.000

Til at flytte containers til oprindelig havn har vi udarbejdet to metoder; plusMinusArray() og flytContainers(). Dette forklares i nedenstående afsnit, da vi har udarbejdet et flowchart af netop disse metoder.

## **Part 4: Flowchart**

Nedenstående flowchart tager udgangspunkt i hvordan containere flyttes tilbage til oprindelig havn. Der er derved taget udgangspunkt i metoderne plusMinusArray() og flytContainers()

### **Flowchart 1: plusMinusArray()**

Flowchart kører igennem et for-loop hvori der forekommer to if-statements. Disse if-statements tjekker om værdien på det givne indeks i saldo array er større eller mindre end 0. Hvis det er større end 0, skal værdien for det givne indeks ligges i sPlus array, på samme indekstal. Derimod, hvis værdien på det givne indeks i saldo array er mindre end 0, skal værdien på det givne indeks i saldo array ligges i sMinus array, på samme indekstal.

```

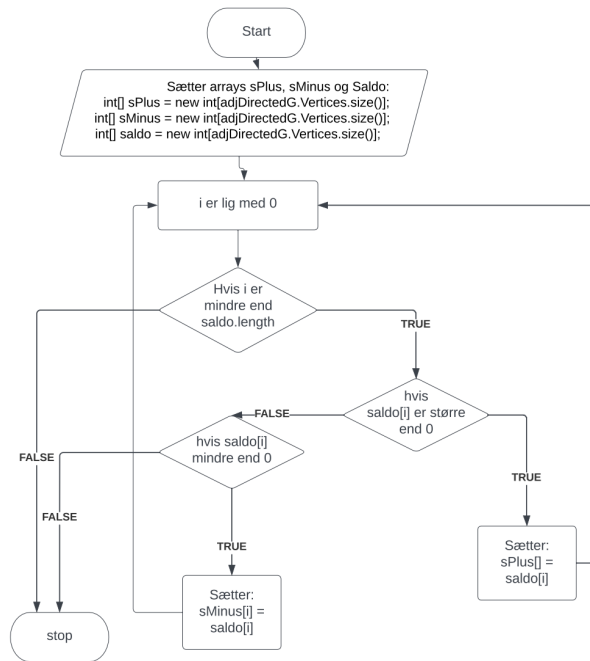
int[] saldo = new int[adjDirectedG.Vertices.size()];
String[] names = new String[adjDirectedG.Vertices.size()];
int[] sPlus = new int[adjDirectedG.Vertices.size()];
int[] sMinus = new int[adjDirectedG.Vertices.size()];
int flytCost = 0;

```

```

public void plusMinusArray() {
    for (int i = 0; i < saldo.length; i++) {
        if (saldo[i] > 0) {
            sPlus[i] = saldo[i];
        }
        if (saldo[i] < 0) {
            sMinus[i] = saldo[i];
        }
    }
}

```



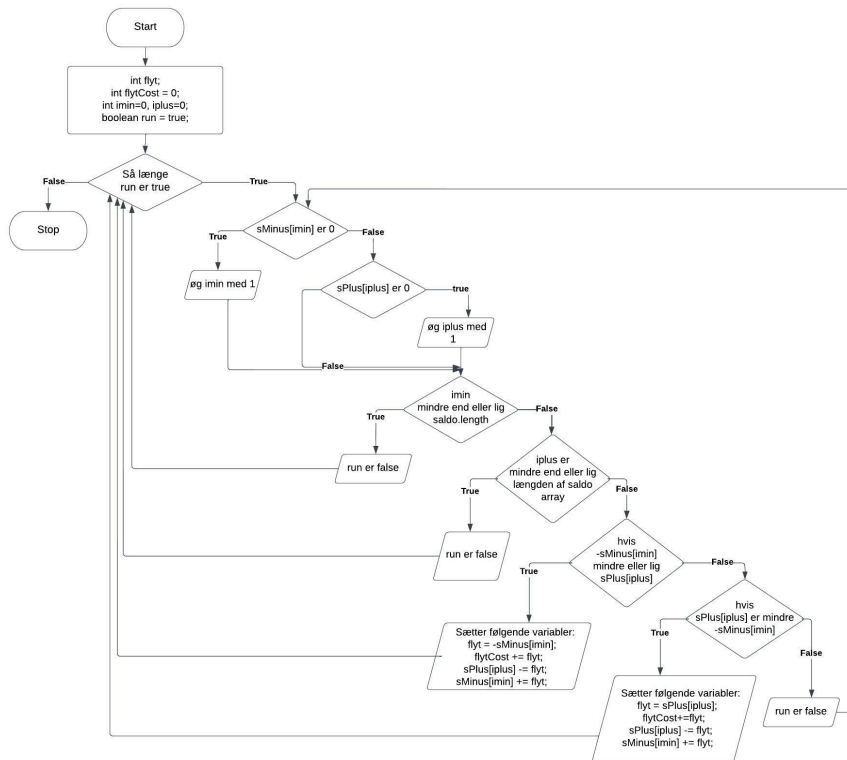
**Flowchart 2:**

Flowchart kører igennem en while-loop, som indeholder fire if-statements og et if-else statement. De fire if-statements tjekker om indekset i sMinus og sPlus er lig med 0, hvis dette er tilfældet, skal den øges. Derudover tjekker if-statementene om imin og iplus er større end eller lig med saldo arraylistens længde. Hvis dette er tilfælde, skal run sættes til false, og while-loopet stoppes. I det efterfølgende if-else statement flyttes containerne, ved at finde det laveste tal der kan flyttes. Hvis værdien af -sMinus[imin] er mindre end eller lig sPlus[iplus], skal værdien for -sMinus flyttes fra sPlus til sMinus. Hvis værdien for sPlus er mindre end -sMinus[imin], skal værdien for sPlus på det givne indeks, flyttes fra sPlus til sMinus.

```

public void flytContainers() {
    int flyt;
    int imin = 0, iplus = 0; //counter
    boolean run = true;
    while (run) {
        if (sMinus[imin] == 0) {
            imin++;
        }
        if (sPlus[iplus] == 0) {
            iplus++;
        }
        if (imin >= saldo.length) {
            run = false;
        }
        if (iplus >= saldo.length) {
            run = false;
            continue;
        }
        //Flytter containerne
        if (-sMinus[imin] <= sPlus[iplus]) { //Tjekker hvor mange der skal flyttes. Finder det laveste tal (ved at bølge værdier i sPlus og sMinus)
            flyt = -sMinus[imin];
            flytCost += flyt;
            System.out.println("Flyt: " + flyt + " fra " + names[iplus] + " til " + names[imin]);
            sPlus[iplus] -= flyt;
            sMinus[imin] += flyt;
        } else if (sPlus[iplus] < -sMinus[imin]) {
            flyt = sPlus[iplus];
            flytCost += flyt;
            System.out.println("Flyt: " + flyt + " fra " + names[iplus] + " til " + names[imin]);
            sPlus[iplus] -= flyt;
            sMinus[imin] += flyt;
        } else {
            run = false;
        }
    }
    printFlytCostPris();
}

```



## Part 5: Komplexitet

I dette program forekommer der bl.a Quadratic Time:  $O(n^2)$ . Dette ses i klassen Flow ved metoden runVertexAndEdges(), hvor der anvendes IndexOf(). IndexOf() indeholder et skjult for-each loop, som lineært gennemløber alle vertices. IndexOf() anvendes i et eget programmeret for-each loop, som også gennemløber Vertices for grafen. Det eget

programmeret for-each loop har n iterationer og indexOf() har også n iterationer, men der forekommer også n iterationer for hver iteration det eget programmeret for-each loop gennemløbes. Det vil sige at for hver vertices gennemløber vi hver vertices. Programmet udfører dermed 100 handlinger pr. 10 elementer.

## **Part 6: Cost forskel ved algoritme og reversing flow**

Reversing: 40.500 overskydende containers \* \$100 = \$4.050.000

Minimal Cost: 25.000 overskydende containers \* 100 = \$2.500.000

Cost forskel: \$1.550.000

## **Part 7: Datastruktur**

Dette program anvender Graph, Arrays og ArrayList til strukturering af data.

Graph datastruktur anvendes til implementering af havne og flowet af containere. Vi har anvendt en directed Graph, hvor Vertices repræsenterer havne, edges repræsenterer flows, og hvor edges har en vægt. Denne vægt repræsenterer antal containere der bliver flyttet fra det ene Vertices (Havn) til en anden Vertices (Havn). Vi har anvendt directed Graph, da flowet, hvor containere bliver flyttet, er fra en bestemt startdestination til en bestemt slutdestination.

Derudover anvendes Arrays som er følgende; Saldo, Name, sPlus, sMinus. Saldo array, indeholder antal overskydende eller antal manglende containere per havn. Name indeholder navnene på havene (Vertices). sPlus array indeholder overskydende containere, og sMinus indeholder havne hvori der er underskud af containere.

Alternativt kunne vi have anvendt Hashmap til at strukturere dataen. Ved at anvende HashMap, ville vi have større kontrol over hvilke havne der har hvilke saldoer, da værdien ville blive gemt i hver deres objekt (Keys og Values). Derudover sikrer HashMap at der ikke forekommer duplikationer (Goodrich et al., 2014).

# Portefølje 3

I denne opgave anvendes kode fra Mads Rosendahl, som er udleveret i forbindelse med kurset Software Development. Koden er udarbejdet ifm. forelæsning “JavaFX Programmering”

## Part 1: Validering

Til at validere databasens integritet har vi justeret i den udleveret database. Vi har tilføjet en række, i flow tabellen, hvor antal containers er større end kapacitet for vessel. Derudover har vi tilføjet en række, i transport tabellen, som har et vid (Vessel id) der allerede forekommer i tabellen. Disse to rækker er ukommenteret i tekstfilen “DatabaseEksamen”. Ydermere har vi ændret attribut navnet, for alle id attributter, således at attribut id’et for Vessel tabellen hedder vid, attributten id for transport tabellen hedder tid, og ligeledes med attributten id for harbour hedder hid (Watt, 2014).

Til at validere at ingen Vessels har flere containers end maksimum kapacitet, har vi udarbejdet en query, hvilket ses på nedenstående billede. Denne query viser en tabel over de flows, hvor summeret antal containere er større eller lig med Vessels maks kapacitet. Tabellen anvender SQL Select statement, til at hente angivet informationer fra Transport tabellen. De ønskede attributter der ønskes hentet forekommer efter “select”. I denne select statement ønskes der information fra de andre tabeller i database, dette muliggøres ved inner join og left outer join til de resterende tabeller. Ydermere forekommer der group by og having. Group by, gruppere sæt af rækker fra Transport tabellen, baseret på at tid har ens værdier. Having filtrer rækker ud, som ikke opfylder betingelsen i parenteserne (“The SELECT Command,” 2010).

```
--Tabel som viser antalContainers fra Flow er større end den specifikke Vessels capacity (Opgave 1)
select t.tid as TransportID, f.fid as FlowID, Sum(f.containers) as FlowContainers, v.capacity as VesselCapacity
from transport t
    inner join vessel v on t.tid = v.vid
    inner join harbour fromharbour on t.fromharbour = fromharbour.hid
    inner join harbour toharbour on t.toharbour = toharbour.hid
    left outer join flow f on t.tid = f.transport
group by t.tid
having (FlowContainers>=v.capacity);
```

Til at validere at ingen Vessels har flere transport på samme dag, har vi udarbejdet en query, hvilket ses på nedenstående billede. Denne query udskriver en tabel, som har 2 kolonner med

kolonne navnene “VesselName” og “antalTransport”. Hvis tilfældet er, at ingen Vessels har flere transport på samme dag, vil tabellen være tom, hvorimod hvis en Vessel har flere transport på samme dag, vil Vessel navnet stå under VesselName kolonnen og det antal transport den har på samme dag, vil fremgå under antalTransport kolonnen.

For at udarbejde denne tabel, har vi anvendt SQL Select statement til at hente Vessels navnet(v.name) fra Transport tabellen samt anvendt Count(), der tæller hvor mange rækker der findes med det specifikke tid i Transport tabellen. Derudover har vi anvendt Inner Join, hvor vi matcher vessel fra Transport tabellen(t.vessel) med vid fra Vessel tabellen(v.vid). Ved at anvende Group By og Having, kan vi sortere vessel fra Transport tabellen, hvor vi kun udskriver Vessels der har flere end én transport på samme dag (Watt, 2014).

```
--Tabel tjekker at ingen vessel har flere end 1 transport om dagen
select v.name as VesselName, count(tid) as antalTransport
from transport t --Tjekker fra Transport, da det er i denne tabel der kan forekomme flere ruter pr. dag
    inner join vessel v on v.vid = t.vessel
group by t.vessel --Fra transport tabel Vessel (VID) er foreign key i Transport tabel
having antalTransport > 1;
```

Udover ovenstående tjek af databases integritet, kan man tjekke hvorvidt Attributten Capacity i tabellen Vessel er null. En vessels kapacitet bør ikke være null. Ydermere kan man tjekke hvorvidt der forekommer duplikationer af tabellernes primary key. En primary key bør ikke være null eller fremgår i flere rækker i tabellen. Derudover kan man tjekke hvorvidt tabellerne har en mange til mange relation, hvis dette er tilfældet, kan der forekomme duplikationer (Watt, 2014).

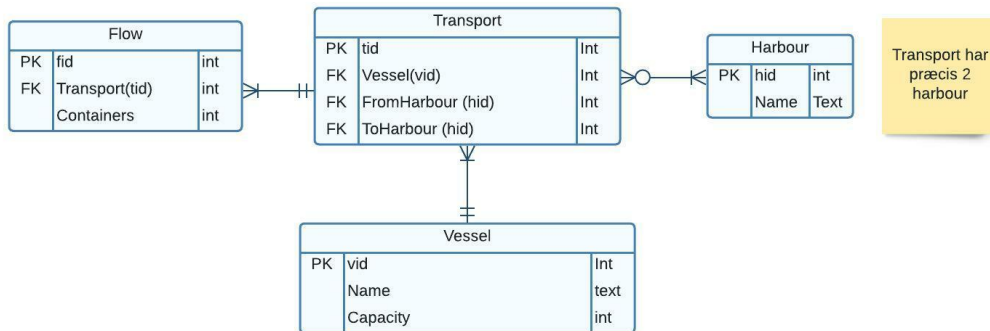
## Part 2: E/R Diagram

Databasen i denne opgave er en relationel database, der forekommer altså relation mellem tabellerne. Denne database indeholder 4 tabeller; Transport, Flow, Harbour og Vessel. Databasen er repræsenteret i et E/R diagram, som kan ses på nedenstående billede.

Hver tabel har en primary key, hvilket illustreres ved “PK”. Primary Key forekommer i en attribut, hvoraf entity skal være unik for hver række i tabellen. Yderligere forekommer der foreign keys, som er en attribut i en tabel, hvoraf attributtens entity er primary key i en anden tabel (Watt, 2014). Relationerne imellem tabellerne er illustreret ved “crow’s feet notation” (Watt, 2014). Crows feet notation viser “cardinality” og “connectivity” mellem tabellerne. Cardinality beskriver multiplicitet fra tabellen og connectivity beskriver multipliciteten til tabellen. En transport kan have præcis to havne, dette er angivet som 1 til flere, da crow feet



notation ikke tilbyder præcis 2. En havn kan have 0 til flere transport. En transport kan have præcis 1 vessel, og en vessel kan have en til flere transport. Et flow kan have præcis en transport, og transport kan have en til flere flow.



### Part 3: User Interface

Vores program’s user interface består af tre Labels, to ComboBox, et TextField og en Button. De to første Labels er hardcoded tekst som angiver hvad ComboBox indeholder, dermed en liste af havne hvor en bruger kan vælge en fra havn og til havn. Den tredje label udskriver “Antal af containers”, som henviser til at brugeren skal skrive et antal containers i TextField. Søgeknappen (Button) bruges til at søge efter de specifikke angivne kriterier.

```

//Fra havn combobox
Label lab1 = new Label( s: "Fra Havn");
ComboBox<String> comboFromHarbour = new ComboBox<>();
for(String fromHarbour: liste){
    comboFromHarbour.getItems().add(fromHarbour);
}

//Til havn combobox
Label lab2 = new Label( s: "Til Havn");
ComboBox<String> comboToHarbour = new ComboBox<>();
for(String toHarbour: liste){
    comboToHarbour.getItems().add(toHarbour);
}
comboToHarbour.getValue();

//Antal containers tekstfelt
Label lab3 = new Label( s: "Antal af containers");
TextField antalContainers = new TextField();

//Søge knap
Button srch = new Button( s: "Søg");
        
```

For at få udskrevet listerne med havnene i de to forskellige ComboBox, definere vi ComboBox som et array med typen String. Derefter udskriver vi listen ved brug af et

for-each loop, der kører alle havne igennem i en arrayliste navngivet "liste". I for-each loopet tilføjes hver havn til comboToHavour, ved at bruge funktionen "getItems()" og "add()".

Arraylisten "liste" er erklæret øverst i metoden start() og anvender referencen "model", og kalder metoden "readListOfHavourNames()". I denne metode, forekommer et sql query gennem en metode "query" som bliver kaldt af "db" som er en instans af MyDB klassen.

Dermed er det inde i MyDB klassen, at forbindelsen til databasen bliver oprettet. Arraylisten "liste" samt metoden i Model "readListOfHavourNames()" kan ses på billederne forneden:

```
ArrayList<String> liste = new ArrayList<>(model.readListOfHavourNames());
```

```
MyDB db=new MyDB();

//Henter liste af havour til combobox query
ArrayList<String> readListOfHavourNames() {
    return db.query( query: "SELECT name FROM havour;", fld: "name");
}
```

Dermed bliver informationerne hentet fra databasen, hvor brugeren ud fra ComboBox kan se listen af navnene på havnene.

Endvidere kalder søgeknappen en eventlister som kan ses på billedet forneden. Dettets gøres, da vi ønsker at brugerens angivne værdier, skal sendes videre til klassen Controller. Controller klassens opgave er at videresende brugerens input data videre til Model, samt at sende et response tilbage til brugeren. Controller anvender "view.setArea()", for at fremvise output til brugeren.

```
srch.setOnAction(e -> controller.search(comboFromHavour.getValue(),
    comboToHavour.getValue(),
    antalContainers.getText()));
```

## Part 4: JavaFX og Database

Vi har tilsluttet vores database til vores java program. Første step for at kunne gøre dette, har for os været at downloade en sql jar file, som vi efterfølgende har tilføjet til vores java projekt. Vi har importeret java.sql.\*; for at kunne tilsutte vores program til databasen.

Vi har lavet en klasse MyDB, hvor vi gennem metode connect tilsluttet databasen, hvor vi har initialiseret conn som værende null.

```
Connection conn = null;
```

I den anden blok har vi brugt en URL string, som går ind og connecter programmet til databasen. Dette er en url string, da den identificerer databasen og dens navn. Vi henter vores

connection objekt ved hjælp af vores DriverManager.getConnection(url). Denne linje forsøger at oprette en forbindelse, idet at den tildeles den tidligere conn variabel. Dette gør at vi har mulighed for at oprette statement forekomster, hvor der printes til konsollen om hvorvidt der er oprettet forbindelse eller ej.

```
public void open(){
    try {
        String url = "jdbc:sqlite:identifier.sqlite";
        conn = DriverManager.getConnection(url);
        System.out.println("Åbner");
    } catch (SQLException e) {
        System.out.println("Kan ikke åbne");
        if (conn != null) close();
    }
}
```

I MyDB klassen har vi lavet en open og en close metode, hvor vi gennem try, catch har mulighed for at se om der er fejl i tilslutningen, eller om databasen tilsluttes vores program korrekt. Når vi har oprettet forbindelse til vores database, har vi mulighed for at trække data omkring de forskellige havne og i databasen, søge efter hvilke vessels der har ledig kapacitet. Søgefunktionen bliver ved at have tilsluttet sit java program til databasen, derfor til at foregå i SQL.

Når brugeren vælger en fra havn, til havn og antal containers hentes en liste ved funktionen readSearchVessel(). Denne funktion tager comboFromHarbour, comboToHarbour og antalContainers som parameter. Funktionen indeholder en SQL query, som anvender funktionens parameter. Denne query vil returnere ledig Vessel som har et flow med den angivne fra- og til havn samt at antallet af containers er mindre end vessels kapacitet.

```
ArrayList<String> readSearchVessel(String comboFromHarbour, String comboToHarbour, String antalContainers){
    return db.query(
        query: "select t.tid as TransportID, fromHarbour.name as fromport, toHarbour.name as toport,"
        + "v.name as vesselName, Sum(f.containers) as antalContainer, v.capacity as containerCapacity"
        + " from transport t "
        + " inner join vessel v on t.vessel = v.vid "
        + " inner join harbour fromHarbour on t.fromharbour = fromHarbour.hid "
        + " inner join harbour toHarbour on t.toharbour = toHarbour.hid "
        + " left outer join flow f on t.tid = f.transport "
        + " where fromport = '" + comboFromHarbour + "' and toport = '" + comboToHarbour + "' "
        + "group by t.tid "
        + " having antalContainer + "+antalContainers+" <= v.capacity ", fId: "TransportID");
}
```

## Part 5: Tilføj cargo

Som nedenstående viser, har vi implementeret programmet således, at når vi kører programmet, har vi mulighed for at tilføje cargo til de forskellige vessel-typer. Dette ses ved nedenstående funktion: addExtraFlow()

```

//Tilføjer ny kolonne til flowtabel med transportid og antalcontainers til flow der generere et nytflow id automatisk
1 usage  ▲ LauraLSJN
void addExtraFlow (String transportID, String antalContainers){
    db.cmd( sql: "INSERT INTO flow(transport,containers) VALUES (" + transportID + "," + antalContainers+");");
}

```

Gennem vores addExtraFlow, når vi tilføjer mere cargo til vores vessels, tilføjes en ny kolonne i vores flowtabel i databasen, hvor at transport ID'et og antallet af containers tilføjes og et nyt flow ID bliver genereret i databasen.

Derudover tjekker programmet jævnfør part 1 i Portfolie 3, de enkelte vessels kapacitet, så der ikke er mulighed for loade mere cargo på de forskellige Vessels, end maks kapaciteten.

Først har vi lavet en Arrayliste i vores metode readSearchVessel(), hvor der ud fra bruger inputtet, søges efter Vessels med ledig kapacitet fra en specifik havn til en anden, samt om der er ledig kapacitet. Derefter en SQL query, i vores metode SearchTransport() som går ind og henter navnet på den vessel som sejler det ønskede flow og hvor der er plads til det angivne antal containere. Hvis det pågældende container antal, plus antallet af containere indtastet af brugeren, er lavere end maks kapaciteten, bliver det muliggjort at tilføje antallet af containere til vesslen.

## Part 6: Udvidelse af database

Den nuværende database indeholder kun information om Transport for en dag. Dette betyder at man ikke kan anvende havne som transit. Freight Department kan derved kun lave et flow, hvis der allerede sejler en Vessel fra det ønskede start- og sluthavn med ledig kapacitet. Databasen kan derved udvides således at det er muligt at gemme information om transport for fremtidige transport.

Databasen kan udvides på forskellige måder. Førstkommende løsninger, er at tilføje to attributter (kolonne), til tabellen flow, hvor den ene attribute er datoen for flowet og den anden attribute indeholder tidspunkt for flow. Derved blive det muligt at søge på fremtidige flows, samt at filtrerer på specifikke datoer.

# Litteraturliste

Anderson, D. J., & Carmichael, A. (2016). *Essential kanban condensed*. Lean-Kanban University.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data structures and algorithms in java* (pp. 60–91). John Wiley & Sons.

Kreibich, J. (2010). *Using sqlite*. “O’Reilly Media, Inc.”

Seidl, M., Scholz, M., Huemer, C., & Kappel, G. (2015). *UML @ classroom: An introduction to object-oriented modeling*. Springer.

The SELECT Command. (2010). In *Using SQLite*. “O’Reilly Media, Inc.”

Watt, A. (2014). *Database Design - 2nd Edition* (2nd ed.). BCcampus.

<https://open.umn.edu/opentextbooks/textbooks/354>