

Grundlæggende Programmering E2020

# Obligatorisk opgave C

## Cargo Ship Booking

### Introduktion

Denne opgave skal laves og afleveres alene, men det er tilladt at snakke sammen med medstuderende, når du laver opgaverne. Spørgsmålene, der skal besvares skriftligt, skal skrives i enten .txt eller .pdf filformat. BLUEJ-projektet og skriftlige besvarelser skal afleveres i én samlet .zip-fil på LEARNIT.

### Opgave 1 - Unchecked Exceptions

Projektet `CargoShipBooking` har fire klasser: `CargoShip`, `Container`, `Booking` og `Demo`. Klasserne er en del af et program, som kan bruges på et fragtskib. Programmet er dog stadig i en tidlig udviklingsfase, så du vil i de næste fire opgaver gøre det mere robust overfor fejl, samt udvide programmet med nye funktionaliteter.

- 1.1 Få et overblik over klasserne `CargoShip`, `Container` og `Booking`, så du forstår, hvad programmet kan indtil videre.
- 1.2 Kør `opg1()`-metoden i `Demo`. Du kommer til at støde på to exceptions, som stopper programmet. Brug `try-catch` kontrolstrukturer til at håndtere dem. Du skal altså *gribe* den specifikke exception, der opstår og udskrive en fejlbesked til brugeren. Du skal altså **ikke** ændre koden i `Demo`-klassen. Når du har håndteret de to exceptions, vil dine to fejlbeskeder blive udskrevet i terminalen, når du kører `opg1()`.

### Opgave 2 - Checked Exceptions

Systemet er nu blevet en anelse mere robust overfor fejl, men der er stadig problemer. Du skal i de følgende opgaver bruge *checked exceptions* til at sikre systemet yderligere.

- 2.1 Konstruktøren i `Booking` tager `int kg` som parameter. Systemet tager på nuværende tidspunkt ikke højde for, at man kan oprette en `Booking` på 0 kg. Opret en ny klasse `InvalidKgException`. Klassen skal i sin konstruktør kalde sin superklasses konstruktør med en fejlbesked som parameter.
- 2.2 Konstruktøren i `Booking` skal nu tjekke, om `kg` er større end 0 før den opretter objektet. Hvis *ikke* `kg` er større end 0, skal konstruktøren *kaste* en `InvalidKgException`.
- 2.3 Vi skal sørge for, at den exception vi muligvis *kaster* også bliver *grebet* igen. `Booking`-konstruktøren bliver kaldt fra `addBooking(...)` i `CargoShip` og dette er derfor et oplagt sted at *gribe* den. Tilføj et `catch`-statement til `addBooking(...)`, der *griber* vores `InvalidKgException` og udskriver fejlbeskeden. Brug metoden `getMessage()` til at udskrive fejlbeskeden.
- 2.4 Systemet har også et andet problem: `addBooking(...)` tjekker på nuværende tidspunkt ikke, om den container, der bliver forespurgt allerede er booket. Opret en ny klasse `ContainerAlreadyBookedException`. Klassen skal have to felter: `int column` og `int row`. Konstruktøren skal tage imod to argumenter: `int requestedColumn` og `int requestedRow`. Derudover skal den kalde sin superklasses konstruktør med en fejlbesked som parameter og sætte felterne til at være lig med de to parametre, der bliver givet.
- 2.5 Derudover skal klassen `ContainerAlreadyBookedException` have en metode `String requestedContainer()`, der skal returnere en streng, der indeholder informationerne om den forespurgte container. F.eks. "Requested container: column 12, row 3".

- 2.6 Opret en ny metode i `CargoShip`, der hedder `checkAddBooking(...)`, der tager imod alle de samme parametre som `addBooking(...)`. Metoden skal tjekke, om den forespurgt container ikke i forvejen er booket. Hvis den er fri, skal den kalde `addBooking(...)`, hvis *ikke* skal den kaste `ContainerAlreadyBookedException(...)`.
- 2.7 Metoden `checkAddBooking(...)` kaldes fra metoden `opg2()` i `Demo`-klassen. Det vil derfor give mening, at det er denne metode, der skal *gribe* exceptionen, hvis den bliver *kastet*. Tilføj en `try-catch` til `opg2()`. Hvis metoden *griber* en `ContainerAlreadyBookedException`, skal den udskrive fejlbeskeden og derefter kalde og udskrive `requestedContainer()`-metoden.

## Opgave 3 - Teori om Exceptions

- 3.1 Forestil dig, at du skriver en metode, der skal skrive data ind i en fil på din computer. Du vil gerne fejlsikre dit program og bruger en `try-catch` i din kode. Ville du gribe en *checked* eller *unchecked exception*, hvis filen ikke kan findes?
- 3.2 Forestil dig, at du skriver koden for et spørgeskema, som spørger om brugerens alder. Din kode er ikke helt god, da den ikke tjekker, om brugeren indtaster tal eller bogstaver. Ville en *checked* eller *unchecked exception* opstå, hvis brugeren indtaster bogstaver?
- 3.3 Se på nedenstående kode:

```
try {
    // some code
} catch (Exception e) {
    System.out.println("The program encountered an exception.");
} catch (FileNotFoundException e) {
    System.out.println("You do not have access to this file.");
}
```

Koden kan compile, men der er noget galt i strukturen. Hvad er "fejlen"?

## Opgave 4 - Udvidelser til programmet

- 4.1 En skærm skal vise et visuelt overblik over fragtskibets bookinger. Lav en metode `void showCargoShip()` i klassen `CargoShip`, som udskriver en matrix, der repræsenterer alle containers på fragtskibet. Hvis en container er booket, skal du udskrive "[X]" og hvis containeren er ledig, skal du udskrive "[ ]". Figur 1 er en udskrift fra et fragtskib med fire rækker og tre containers per række, hvor containerne 1,2; 2,2; 3,0; 3,1 er bookede (de samme som i `Demo`).

```
[X] [X] [ ]
[ ] [ ] [X]
[ ] [ ] [X]
[ ] [ ] [ ]
```

Figur 1: Eksempel på udskrevet matrix

- 4.2 Lav en metode `double flammableGoodsFactor()` i klassen `CargoShip`, som udregner et tal, der giver en idé om, hvor brandfarligt skibet bliver. Tallet skal udregnes ved at dividere antallet af bookinger, hvor brandfarlige varer er inkluderet, med det samlede antal containers på fragtskibet. Vær opmærksom på typer i denne opgave.
- 4.3 Kør metoden `opg4()` i `Demo`-klassen, for at teste, at de to metoder virker som forventet. `flammableGoodsFactor()` skal returnere  $\sim 0.1667$ .