Collections, for-løkker, while-løkker

Grundlæggende Programmering
med Projekt

Dan Witzner Hansen

Diverse

- Sidste gang?
- Hvor mange bruger slides efter forelæsningen?

Collections

- Software håndterer ofte en samling af objekter af samme type:
  - En notesbog med noter
  - Et bibliotekskatalog med bøger
  - En studieadministration med kurser
  - En simuleret skov med træer
- Antallet af objekter varierer over tid:
  - Nye tilføjes, gamle slettes
- Til at håndtere samlinger bruger man **collections**, fx ArrayList

Eksempel: Musik

- Ny files(en string) kan tilføjes
- En gammel file kan vises
- Der er ingen grænse på antallet af files

- Java-projekt *MusicOrganizer*

Java-kode for notesbogen

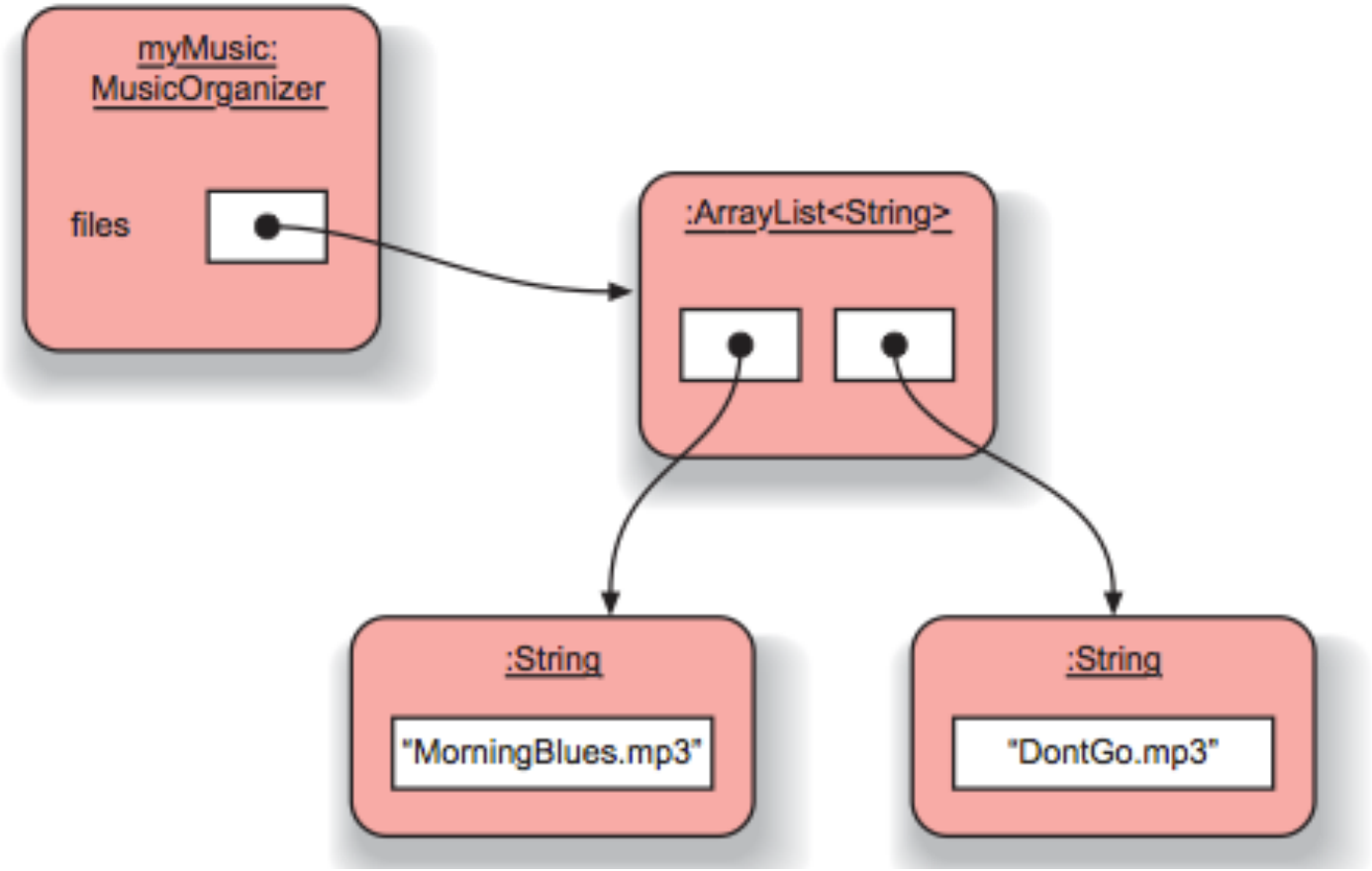
- En MusicOrganizer har et felt **`files`**, som er en arrayliste
- Arraylisten indeholder filerne, som er strings

```
import java.util.ArrayList;

public class MusicOrganizer
{
    private ArrayList<String> files;

... metoder ...
}
```

Objektstruktur for notesbog



Arraylister

- En arrayliste er en *collection* (samling); den kan indeholde mange andre objekter
- En arrayliste indeholdende String-objekter har klasse ArrayList<String>
- En arrayliste oprettes med **`new`**

```
new ArrayList<String>()
```

- Klassen stammer fra pakken java.util i Javas klassebibliotek, derfor

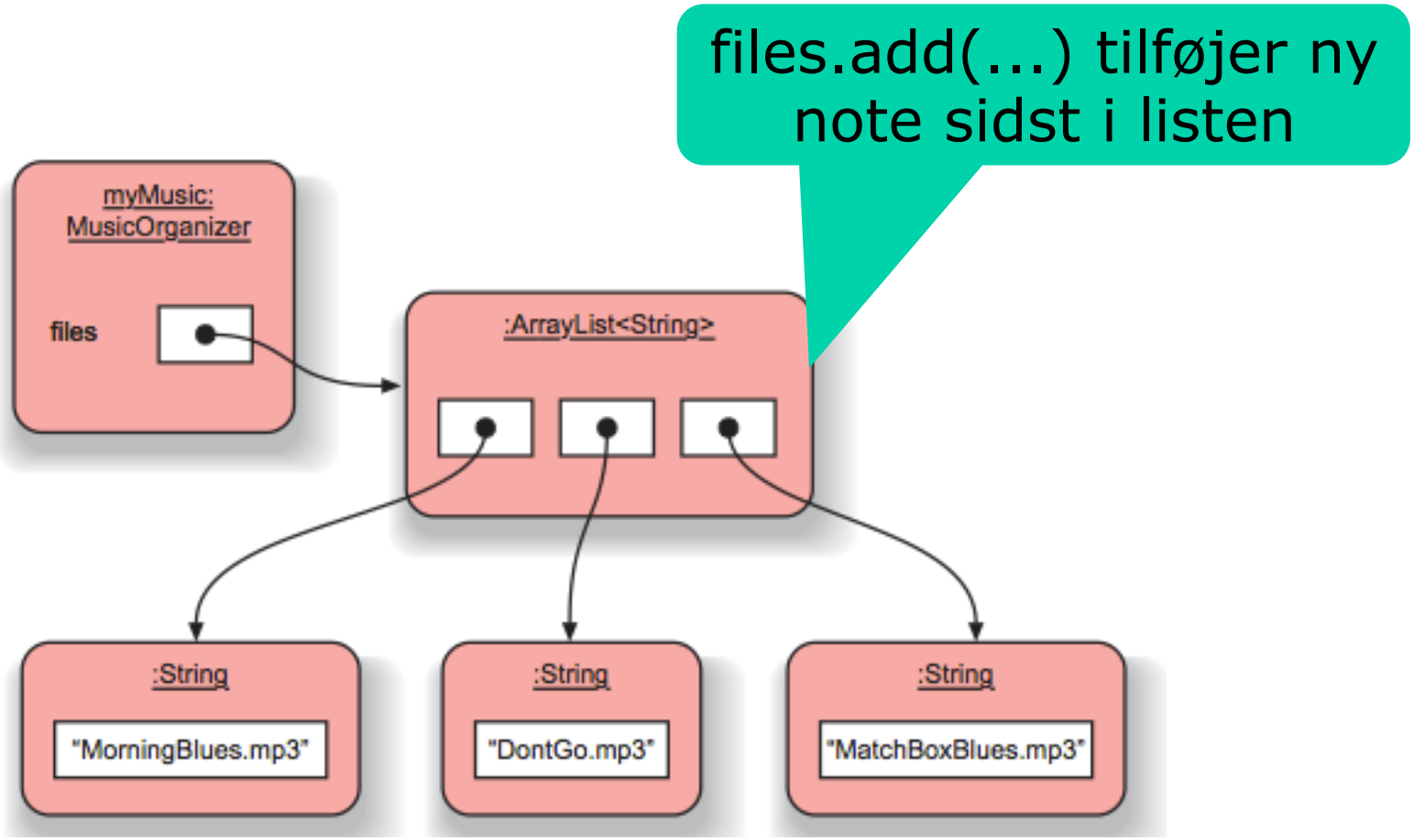
```
import java.util.ArrayList;
```

Tilføj note

- Metoden **`files.add(note)`** tilføjer strengen note sidst i arraylisten
- Det kan bruges til at indsætte noter:

```
public void addFile(String filename)
    {
        files.add(filename);
    }
```

Tilføjelse sker sidst i listen



Generiske klasser

- ArrayList er en generisk klasse:
  "generic: *general, not specific or special"*
- Et objekt af klasse ArrayList<String> kan indeholde String-objekter
- Det læses: *ArrayList of String*
- Et objekt af klasse ArrayList<Tree> kan indeholde – hvad mon?

- Generiske typer er nye (2004) i Java og C#

Indeksering i arrayliste

- Metoden **`files.get(i)`** returnerer element nummer **`i`** fra arraylisten
- Det kan bruges til at genfinde filer

```
public void showFile(int i) {
   System.out.println(files.get(i));
}
```

- Men vi burde have et tjek på at **`i`** er et lovligt indeks i arraylisten ...

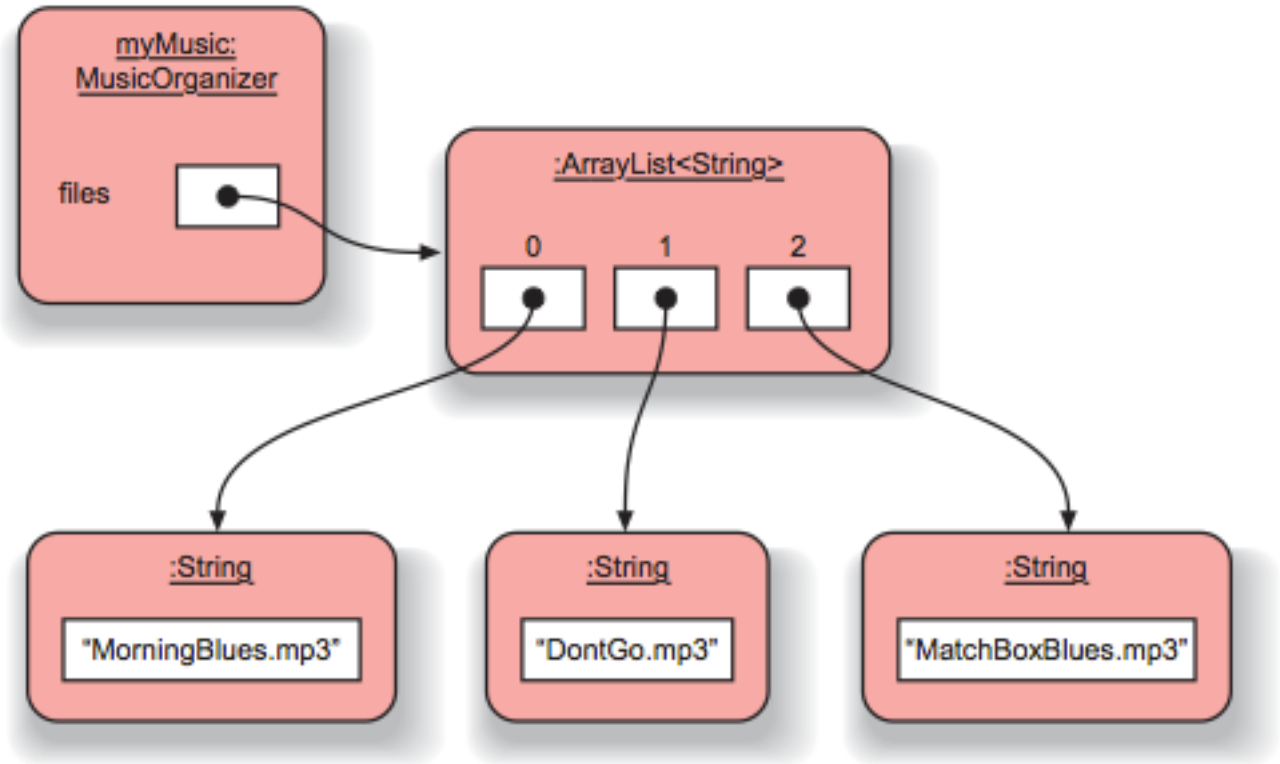
Antallet af elementer

- Metoden **`files.size()`** antallet af elementer i listen

```
public int getNumberOfFiles()
    {
        return files.size();
    }
```

- Hvorfor bruge et metode kald?

Indeksering i arrayliste



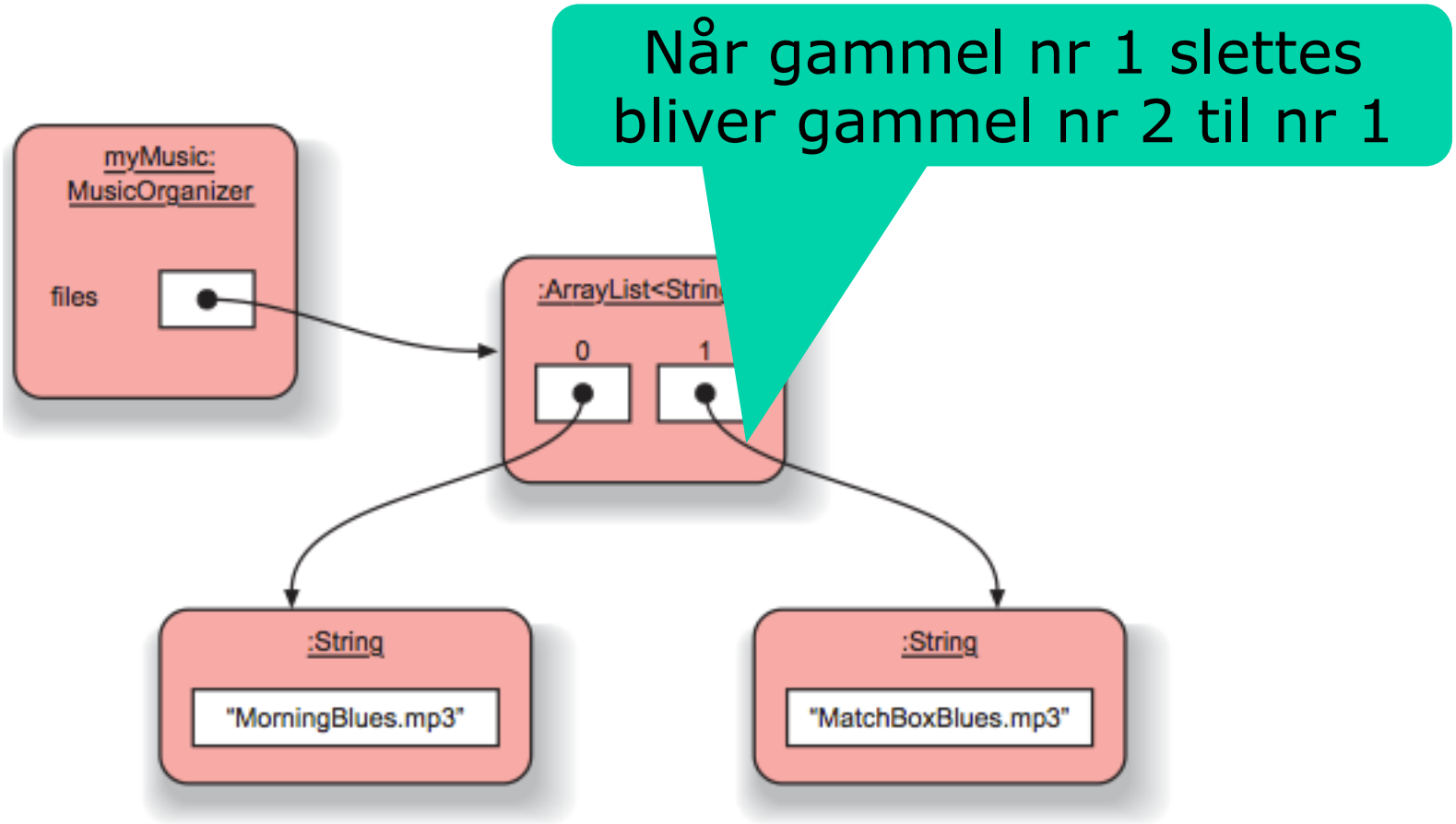
Sletning fra en arrayliste

- Metoden **`files.remove(i)`** fjerner element nummer **`i`** fra listen **`notes`**
- Det kan vi bruge til at programmere sletning af noter i notesbogen:

```
public void removeNote(int i) {
   files.remove(i);
}
```

- Også her burde vi tjekke at **`i`** er et lovligt indeks i arraylisten

Sletning påvirker indekserne



Tjekopgaver

- Løs disse **uden** computer:
  B&K 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8

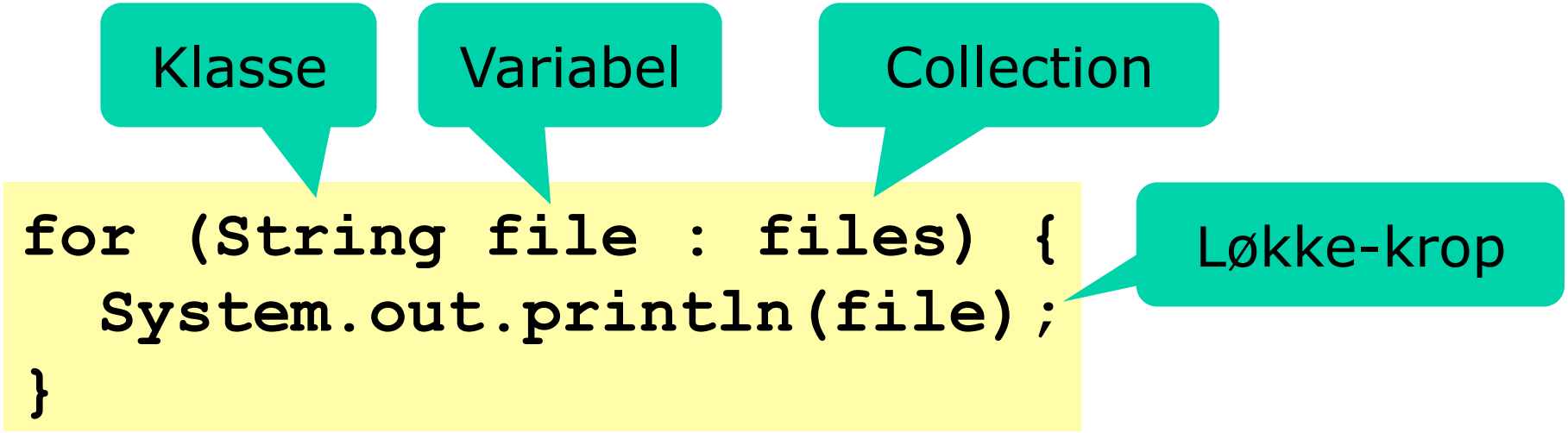
- Brug 10 minutter på opgaverne

Gennemløb af collections

- Hvordan kan vi udskrive alle noter?
  - Vi ved jo ikke hvor mange der er...
- Svar: Brug en foreach-løkke

```
public void listAllFiles() {
   for (String file: files) {
     System.out.println(file);
   }
}
```

Foreach-løkkens opbygning



- Læses: *For each* ***`file`*** *(of class String) in collection* ***`files`****, execute the loop body*

Mere generelle løkker: while

- En while-løkke behøver ingen collection
- Kun en betingelse og en løkke-krop (og en initialisering)
- Eksempel: Skriv tallene 0, 2, 4, ..., 30

```
int number = 0;
while (number <= 30) {
  System.out.println(number);
  number = number + 2;
}
```

While-løkkens opbygning

Betingelse

```
while (number <= 30) {
   System.out.println(number);
   number = number + 2;
}
```

Løkke-krop

- Læses: *While the condition is true, execute the loop body*
- Mere præcist: (1) Udregn betingelse; (2) hvis false, stop; hvis true, udfør løkke-krop og fortsæt med (1)
- NB: Efter løkken er betingelsen falsk

Gennemløb af arraylist med while i stedet for foreach

- Det er muligt men ikke kønt

```
int index = 0;
while (index < numberOfFiles()) {
   System.out.println(files.get(index));
   index++;
}
```

Samme som
index=index+1

Foreach versus while

- Fordele ved ”foreach”-løkke
  - Nemt at gøre rigtigt
  - Kører gennem alle elementer
  - Stopper altid
  - Overskuelig
- Fordele ved while-løkke
  - Meget mere generel
  - Behøver ikke gennemløbe alle elementer
  - Ikke ved antallet af elementer på forhånd
- Farer ved while-løkke
  - Uendelige løkke
  - Kører en gang for langt eller en for kort

While-eksempel med tal

- Mål: Find mindste n sådan at
  1/1 + 1/2 + 1/3 + … + 1/n > 10

```
double sum = 0.0;
int n = 0;
while (sum <= 10)
{
   n++;
   sum = sum + 1.0/n;
}
System.out.println(n);
```

- Er denne løkke korrekt? Hvorfor?

While-eksempel: Søgning i arrayliste

- Mål: Find første note som indeholder ”**superDan**” som delstreng

```
int index=0;
boolean found=false;
while (index<numberOfNotes() && !found) {
  String file= files.get(index);
  if (file.contains(“superDan”)) {
    found = true;
  } else {
    index++;
  }
}
```

- Brug debuggeren til at forstå løkken

U-løkker

- Antag `int x=0` og `int y=0`
- Hvad er der galt nedenfor?

```
while (x >= 0) {
   x++;
}
```

```
while (y <= 10) {
   x++;
}
```

```
while (y <= 10); {
   y++;
}
```

Tjekopgaver

- Lav disse opgaver med computer:
  - B&K 4.9, 4.12, 4.16, 4.21 om MusicOrganizer
  - B&K 4.17, 4.18 om løkker.
    Lav et nyt projekt og en ny klasse til dette formål,
    og lav en metode til hver delopgave

Næste uge

- Fortsætte med Kapitel 4. Denne uge vil være lidt sværere at forstå end tidligere.
- **Brug lidt mere tid på at sætte jer ind I materialet inden forelæsning.**
- Brug også gerne tid på at forstå Auction-eksemplet