



NTNU

Kunnskap for en bedre verden

Vi fortsetter igjen 11:15

**Forelesningen tas opp
automatisk mens det
røde lyset ved kateteret
er på.**



TDT4100 Objektorientert programmering | 09.02.2024

Forelesningen begynner 10:15

**Forelesningen tas opp
automatisk mens det
røde lyset ved kateteret
er på.**





NTNU

Kunnskap for en bedre verden

Øvingsforelesning 3

TDT4100 Objektorientert programmering

09.02.2024

Eirik Lorgen Tanberg

Vitenskapelig assistent, TDT4100

eirik.l.tanberg@ntnu.no



Agenda

- Administrativt
- Repetisjon og litt nytt:
 - Wrapperklasser
 - Vanlige fallgruver for øving 2/3
- Introduksjon til øving 3
- Relevant teori til øving 3
- Praktisk oppgaveløsning
 - Klassediagram

Administrativt

- Håper alle har husket å demonstrere øving 1
- Øving 2 har innleveringsfrist i dag, fredag 9. februar kl. 23:59
- Øving 3 har innleveringsfrist neste uke, fredag 16. februar kl. 23:59

Dat typer og *wrapperklasser*

Wrapperklasser

Integer



Primitiver

int

Double



double

Boolean



boolean

Character



char

String

...

Wrapperklasser gir mer innebygd funksjonalitet enn de primitive datatypene. Se mer på <https://www.ntnu.no/wiki/display/tdt4100/Tall+og+beregninger>

Wrapperklasser

Eksempel

```
public class Integer {
```

```
    private int number = 5
```

```
    public static int parseInt(String s)
```

```
    public static int signum(int i)
```

```
    .  
    .  
    .
```

numberObject

Wrapperklasser

Eksempel

```
public class Integer {
```

```
    private int number = 5
```

```
    public static int parseInt(String s)
```

```
    public static int signum(int i)
```

```
    .  
    .  
    .
```

numberObject

← Wrapperklasse

Wrapperklassen **Integer** inneholder en primitiv **int** som et **felt**

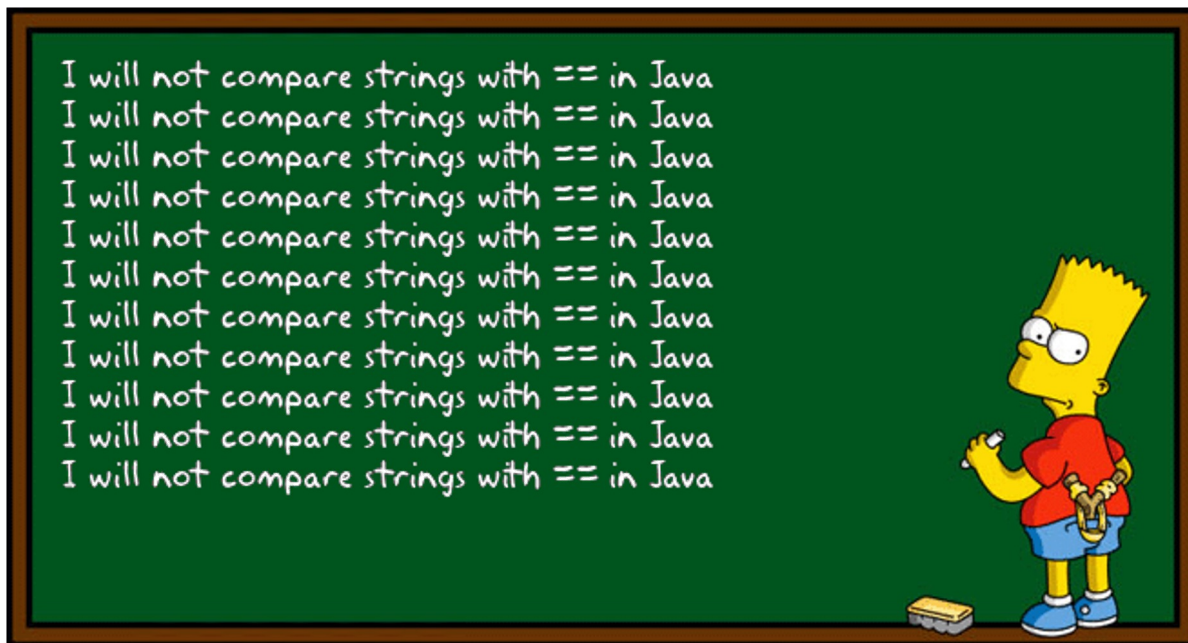
← Nyttige hjelpefunksjoner er inkludert i wrapperklassen!

Eksempel

Forskjellen mellom en primitiv og wrapperklassen

Noen vanlige fallgruver


— Og hvordan man unngår de



Sammenligning av strenger

Hva vil være output av nederste linje i koden her?

- Er det som forventet?
- Eventuelt: Hvorfor ikke?



```
1 String longString = "Dette er en lang streng";  
2 System.out.println(longString.substring(0, 5));  
3 System.out.println(longString.substring(0, 5) == "Dette");
```

Sammenligning av strenger

`streng1 == streng2` Sjekker om begge objektene peker til **samme lokasjon i minnet**

`streng1.equals(streng2)` Evaluerer **verdiene** i objektene, altså den faktiske strengen

Sammenligning av strenger

Ved sammenligning av **objekter**, som inkluderer wrapperklasser, er det derfor viktig å bruke `.equals()`

Ved sammenligning av **primitive datatyper** kan man benytte `==`. Det gjelder for **int**, **double**, **char**, **boolean**, etc.

Eksempel

Vi ser på filen `StrengSammenligning.java`

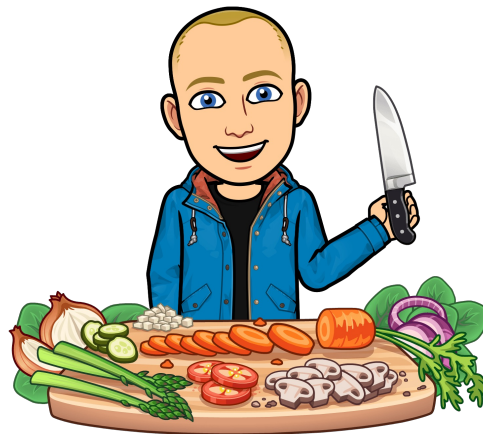
.split() og RegEx

`^(?=.*[0-9])(?=.*[a-zA-Z]).{6,10}$`

- Flere har nok allerede støtt på problemer når man prøver å bruke **`string.split(".")`**
 - Inputparameteret i **`split`**-metoden er et **regex-uttrykk**, og tolkes derfor ikke likt som en vanlig streng.
 - På regex-språket er punktum (`.`) litt som jokeren i en kortstokk: den kan representere alle mulige bokstaver/tall/tegn.
 - Resultatet blir at vi **splitter** på **alle mulige tegn**.
 - For å unngå at Java tolker punktumet vårt slik så må vi **escape** punktumet, noe vi gjør ved å sette to backslasher foran (`\\`)

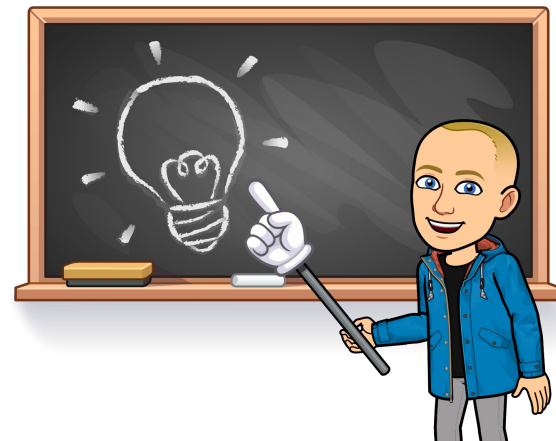
Eksempel

Vi ser på filen `StrengSplitting.java`



Introduksjon til Øving 3

Læringsmål og informasjon



Øving 3

- Tema: **Klasser og debugging**
 - Lære å lage enkle Java-klasser og -programmer
 - Lære å bruke debuggeren i VSCode
- Krav for å få godkjent øvingen:
 - Kunne tegne et enkelt klassediagram
 - Kunne deklarere klasser og metoder ihht. oppgavespesifikasjon
 - Vise grunnleggende forståelse av debuggeren i VSCode
- Leveringsfrist på Blackboard fredag 16. februar kl. 23:59

Øving 3

- Ikke så mange nye konsepter rent kodemessig, men vi stiller nå krav til bruk av **debuggeren**.
- Hittil har vi fokusert på viktige konsepter i objektorientert programmering og mye struktur. Nå skal vi bygge på det vi har lært og legge til **mer funksjonalitet** i koden vår.
- Du skal velge minst én av oppgavene **Nim**, **Card** og **RPN-kalkulator** som du skal kode og tegne et klassediagram for. I tillegg skal du gjøre en obligatorisk oppgave med bruk av debuggeren.
- Card-oppgaven anbefales **sterkt**, hvis man bare gjør **Nim** vil det være tyngre å komme inn i mange av konseptene på øving 4.

Debuggeren

- Viktig generelt verktøy i programmering, brukes for å analysere kjørende kode
- Obligatorisk oppgave på øving 3, **CoffeeCup**
- Egen Wiki-side:

<https://www.ntnu.no/wiki/pages/viewpage.action?pageId=235996724>

- Debugging-funksjonaliteten innebærer å kjøre programmet interaktivt og kunne se verdien i bla. variabler underveis mens man kjører koden.
 - I VSCode: Trykk på **Debug** til høyre for “Run” over main-metode
 - Vha. **breakpoints** i koden kan du spesifisere hvor eksekveringen av programmet skal stoppe



Funksjoner i debuggeren

- **Continue:** Koden kjøres frem til neste breakpoint
- **Stop:** Avslutter kjøringen
- **Step into:** En del av steppe-funksjonen. Går inn i neste blokk med kode. Dersom det er et metodekall på neste linje som skal utføres, vil Step into ta deg inn i denne slik at du kan fortsette debuggingen der.
- **Step over:** Tar deg til neste kodelinje i den filen du er i nå. Dersom du har et metodekall på linjen vil du altså ikke gå inn i denne metoden.
- **Step out:** Dersom du har brukt Step into for å komme inn i en metode vil Step out ta deg ut igjen.



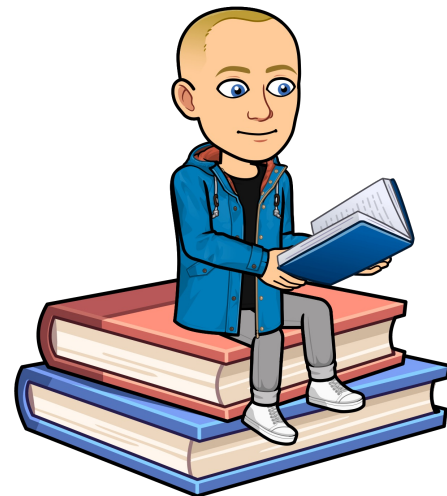
Eksempel

Vi ser kjapt over debuggeren i VSCode



Teori for øving 3

Arrays, lister og statiske felt/metoder



Arrays

```
1 int[] mineTall = {1, 2, 3, 4, 5};  
2 System.out.println(mineTall[0]);
```

- **Arrays** ligner på **lister** som kjent fra Python
 - **Men:** i Java har arrays **statisk størrelse**
 - Når de først er initialisert kan du **ikke** endre størrelse på den senere
- Arrays kan brukes til mye, men ofte er vi avhengig av mer funksjonalitet. Derfor bruker vi i stedet...

ArrayList

- Datastrukturen/klassen **ArrayList**:
`public class ArrayList<E>`
- Definert i **java.util**-pakken (må importeres)



```
1 List<String> someList = new ArrayList<>();
```

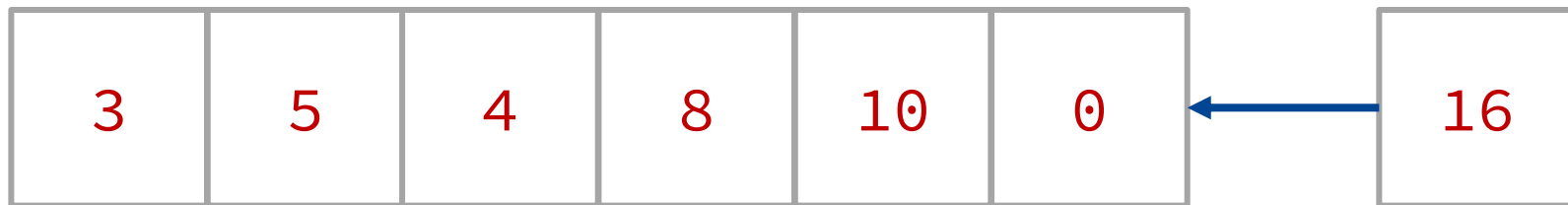
- Som regel **deklarerer** vi **ArrayList** som en **annen type** (eksempelvis **Collection**, **List**).

ArrayList

- Støtter **dynamisk størrelse**, som kan **vokse etter behov** (i motsetning til vanlige arrays).
- Aktuelle metoder i klassen å lære seg:
 - **add(E object)**
 - **get(int index)**
 - **set(E object, int index)**
 - **size()**
- Les mer:
<https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/util/ArrayList.html>

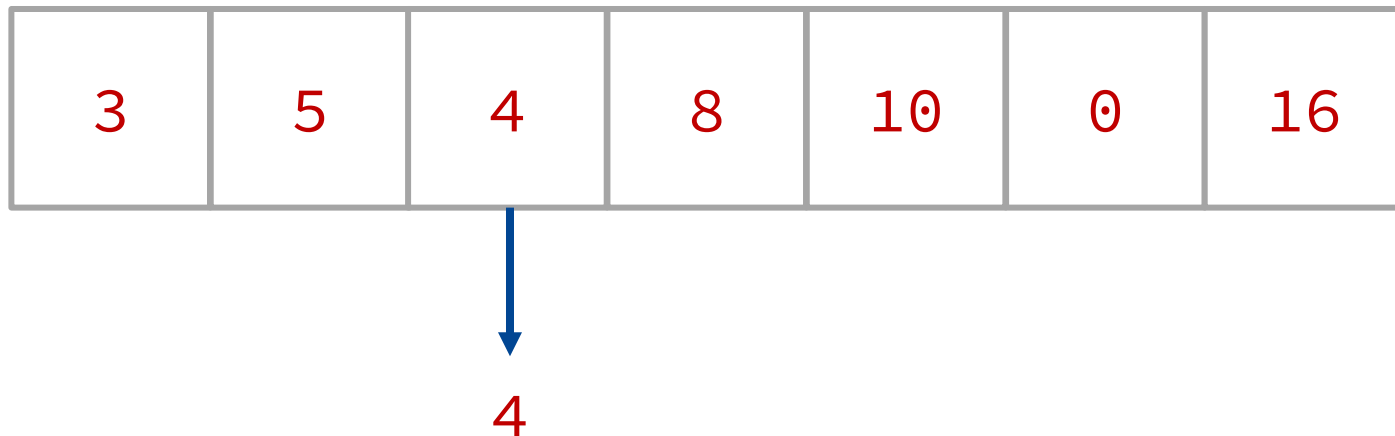
ArrayList

`add(E object)`



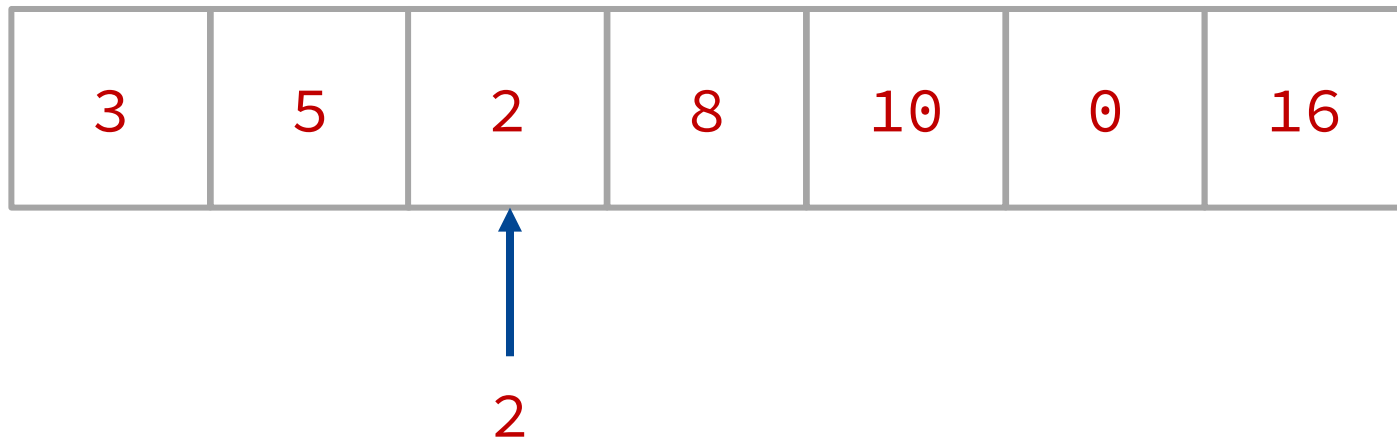
ArrayList

`get(int index)`



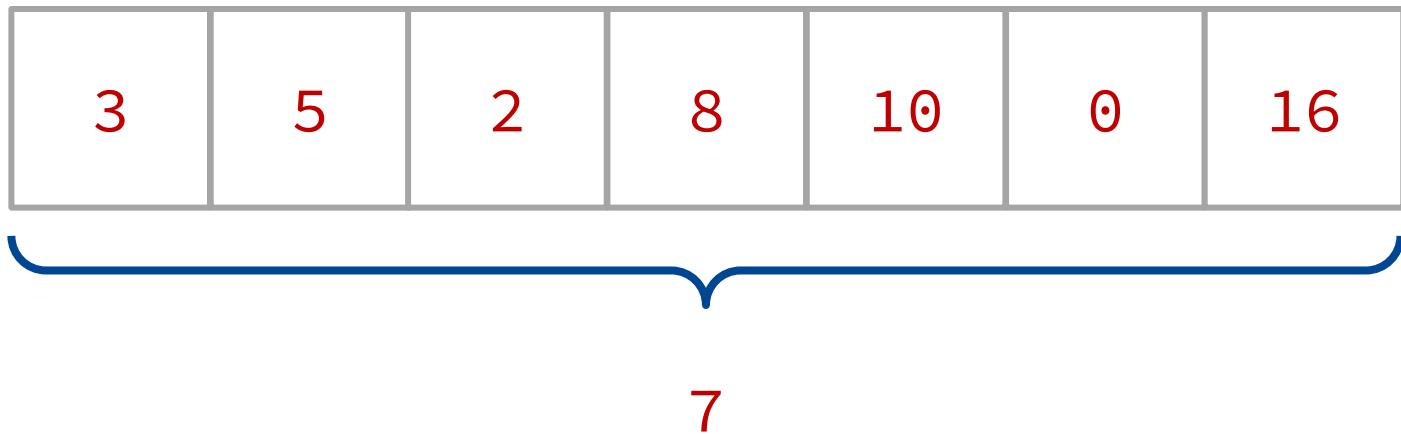
ArrayList

`set(E object, int index)`



ArrayList

size()



ArrayList

Nye datastrukturer: `List<E>`, `ArrayList<E>`

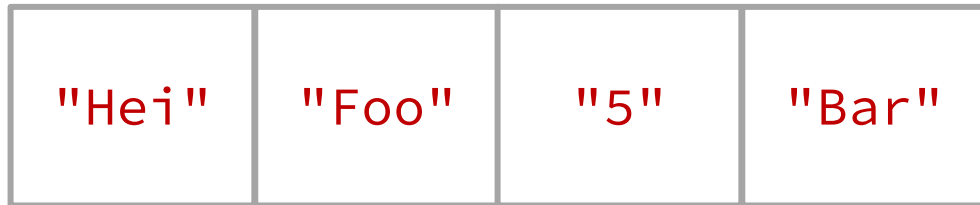
- **E** er en **typeparameter**. Dette angir hva slags **type objekter** datastrukturen inneholder
- I stedet for **E** kan det stå f.eks. **String** eller **Integer**:

NB! Kan ikke være primitiv

`ArrayList<Integer>`



`ArrayList<String>`



ArrayList

Vi deklarerer vanligvis en liste slik:

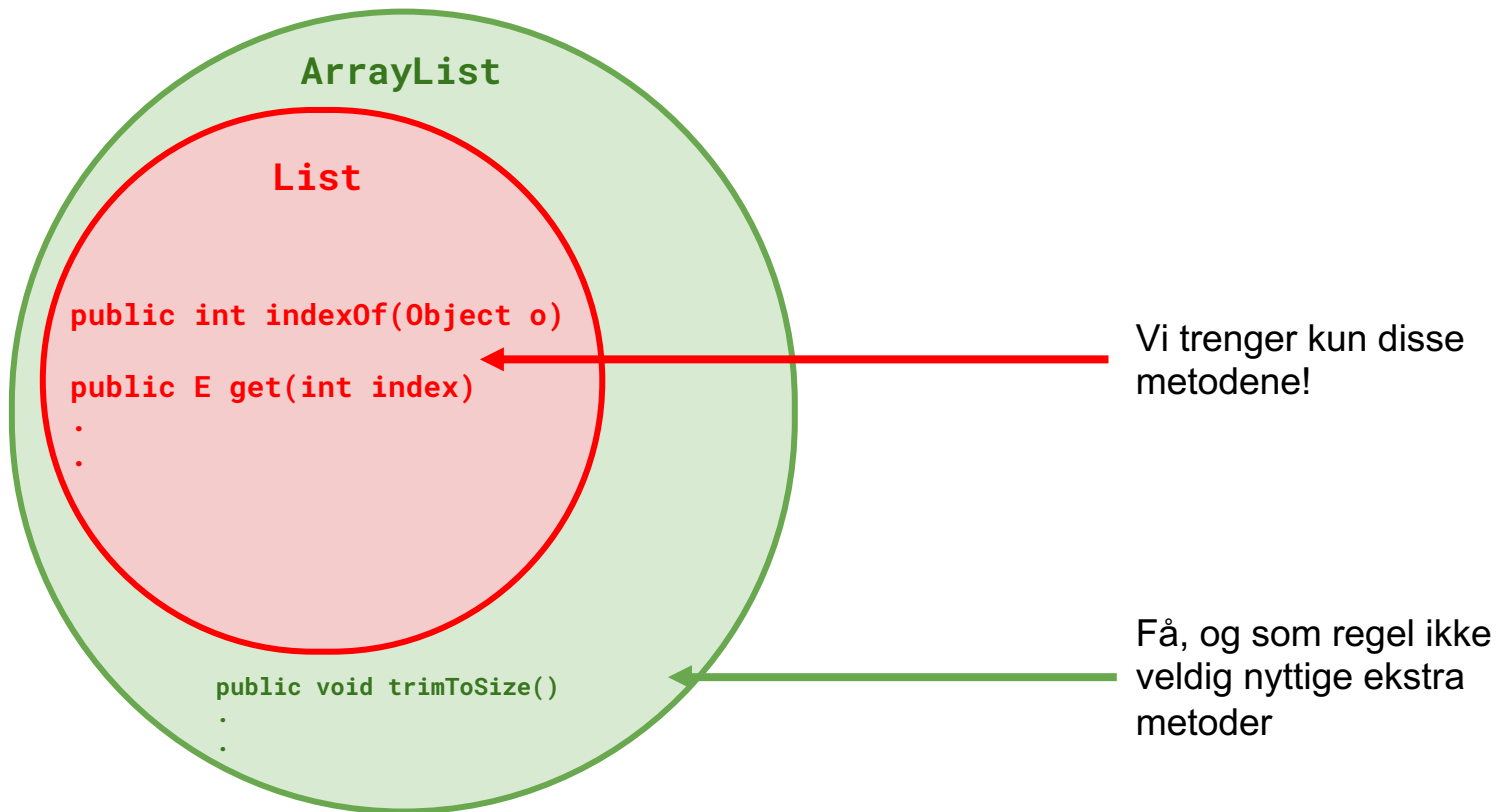
```
List<Object> liste = new ArrayList<>();
```

Merk at vi bruker **List** på **venstre side** og **ArrayList** på **høyre side**. Dette er pga. **arv** og **grensesnitt** som vi lærer om senere i kurset og fordelene vil derfor ikke være umiddelbart åpenbare.

Vanligvis trenger vi kun metodene som er definert i **List**. Det er noen ekstra metoder i **ArrayList** sammenlignet med **List**, men disse får dere neppe bruk for uansett.

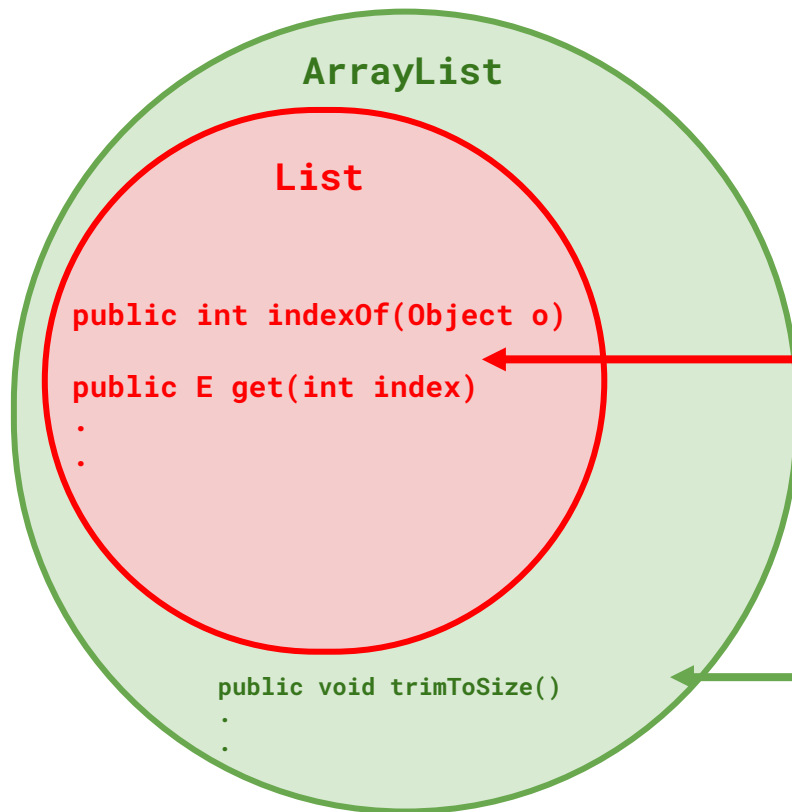
ArrayList vs. List

Visualisering



ArrayList vs. List

Visualisering



Merk: Selv om vi kun trenger metodene som også er i `List` så er **ikke** dette en klasse, og den kan derfor ikke **instansieres**.

Derfor **instansierer** vi `ArrayList`-objekter men **deklarerer** de som `List`-objekter

Vi trenger kun disse metodene!

Få, og som regel ikke veldig nyttige ekstra metoder

Statiske felter og metoder

- Vi bruker dette dersom vi **ikke** trenger å bruke/endre **intern tilstand** inne i en metode, altså “generelle” metoder.
 - Eksempler på **statiske metoder** kan være *hjelpemetoder*, *valideringsmetoder*, *main-metoder*
 - Tenk: Metoder som kun tar inn parametre og returnerer noe (bruker ikke interne felter / tilstand)
- Kan også brukes når vi har variabler/konstanter som er **felles** for alle objekter i en klasse
- Variabelen/metoden tilhører hele klassen, og ikke en spesifikk instans.

```
public static void main(String[] args) {  
    TeslaCar myCar = new TeslaCar("TDT4100");  
}
```

Eksempel

I klassen **SelfServiceCheckout** så finner vi et eksempel på et **statisk felt**.

days er felles for alle instanser, antall dager som finnes totalt vil jo aldri endre seg

Hvilken **spesifikk dag**
det er (day) vil endre seg
for hver gang klassen
instansieres, derfor kan **ikke**
denne være statisk

```
1 public class SelfServiceCheckout {  
2  
3     public static final List<String> DAYS =  
4         Arrays.asList("mon", "tue", "wed", "thu", "fri", "sat", "sun");  
5  
6     private String day;  
7     private String phoneNumber;  
8  
9     // ...  
10 }
```

Oppgaveløsning

SelfServiceCheckout.java og Item.java



Forrige gang (med litt endringer):

Selvbetjent kassaapparat

Det nasjonale butikkonsernet *OOP mini* ønsker å lage nye selvbetjente kassaapparater for å gjøre handlingen enklere og mer effektiv.



Dagens oppgave er å utvikle en **prototype** for dette kassasystemet som kan gjøre enkle operasjoner som å scanne inn varer, regne ut pris / MVA (totalt og per vare), gi rabatt på enkelte varer / dager, samt printe ut en kvittering.



Målsetting

Vi skal få printet ut en fint formatert kvittering med oversikt over kostnadene.

Hva	Pris	MVA	Total
1 x Tomato	4,25	0,75	5,00
2 x Cheese	25,50	4,50	60,00
1 x Burritos	38,25	6,75	45,00
Total MVA			16,50
Total			110,00
Takk for at du handlet hos oss i OOP!			

SelfServiceCheckout - del 1

Vi skal opprette en klasse som representerer den selvbetjente kassen,

Vi utvider **SelfServiceCheckout** fra forrige uke med følgende felter. Utvid konstruktøren til å også ta inn **password** som et parameter, samt initialiser feltene til passende verdier

<code>day</code>	En tekststreng som representerer ukedag
<code>adminMode</code>	En boolsk verdi som viser om kassen er i admin-modus eller ikke
<code>password</code>	En tekststreng som gir passordet for å sette kassen i admin-modus
<code>shoppingCart</code>	En List av typen ArrayList som inneholder item-objekter
<code>days</code>	En såkalt <i>konstant</i> som skal være satt som static og final . Dette gjør at feltet ikke kan endres i etterkant, og vil være tilgjengelig uten å instansiere et objekt. Feltet skal inneholde en liste med strenger av dager på formatet mon, tue, wed, thu, fri, sat, sun
<code>phoneNumber</code>	Et telefonnummer som kunder skal kunne skrive inn for å eksempelvis få rabatter eller tilsendt kvittering på mobil.

SelfServiceCheckout - del 2

a) Lag metoden **validatePassword** som sjekker at et passord (streng) tilfredsstiller følgende krav og **bruk denne i konstruktøren**:

- Passordet skal ha mellom 6 og 10 tegn
- Passordet må inneholde minst ett tall (0-9) og minst én bokstav fra det engelske alfabetet (A-Z).

Benytt følgende **RegEx**:

```
^(?=.*[0-9])(?=.*[a-zA-Z]).{6,10}$
```

b) Opprett setter for **adminMode**. Metoden skal ta inn et passord og sette feltet **adminMode** til **true** dersom passordet er riktig, og kaste unntaket **IllegalArgumentException** hvis ikke. I tillegg skal **IllegalStateException** utløses hvis man prøver å sette adminMode **på nytt**.

IllegalArgumentException vs. IllegalStateException

- Dersom argumentet (input) til funksjonen er **ugyldig** eller vil gjøre at klassen havner i en **ugyldig tilstand** så skal **IllegalArgumentException** utløses:

```
if (!days.contains(day)) {  
    throw new IllegalArgumentException("Invalid weekday");  
}
```

- Dersom en metode brukes på et **ulovlig** tidspunkt, eller når objektet er i en tilstand hvor det ikke gir mening å bruke metoden, så skal **IllegalStateException** utløses:

```
if (this.adminMode) {  
    throw new IllegalStateException("Admin mode is already active!");  
}
```



Item

Vi har allerede opprettet en **Item**-klasse som inneholder feltene **name** (string), **price** (double), **category** (string) og **barcode** (string) som representerer en vare man kan handle i butikken.

Item er en såkalt **dataklasse** som vil si at den i tillegg til datafelter og konstruktør, kun inneholder enkle metoder for å aksessere og endre informasjon, i form av **getters** og **setters**. Dette vil si at klassen ikke inneholder noe funksjonalitet, slik som å regne ut pris eller lignende.

SelfServiceCheckout - del 3

- a) Vi skal nå lage funksjonen **scanItem** som tar inn et **Item**. Funksjonen skal legge til denne varen i shoppingCart, og printe ut informasjon om varen på følgende format:

<varenavn>: <pris> kr

- b) Opprett metoden **scanItems** som tar inn en liste med **Item**-objekter, itererer gjennom den og scanner alle Item-objektene ved hjelp av **scanItem**-funksjonen fra oppgave a)

SelfServiceCheckout - del 4

- a) Opprett en metode kalt **removeFromCart** som tar inn **indeksen** til en vare som skal fjernes fra handlekurven. Dersom **adminMode** ikke er satt til **true** skal metoden utløse et **IllegalStateException** da kun butikkansatte skal kunne fjerne varer fra handlekurven til kunder.
- b) Opprett metoden **isMember** som sjekker om kunden har fylt inn et mobilnummer eller ikke og returnerer en **boolean** deretter.

SelfServiceCheckout - del 5

OOP mini er store på taco, og ønsker å gi alle **medlemmene** sine fast rabatt på tacoingredienser i helga

a) Opprett metoden **getDiscountForItem** som returnerer en rabatt for en vare, eventuelt bare **0.0** dersom det ikke er noen aktuell rabatt på varen. Dersom det er **fredag** eller **lørdag** og **kategorien** til **Item**-objektet er **"taco"** skal kassen gi 30% rabatt.

b) Lag deretter metoden **getPriceForItem** som returnerer prisen på en gitt vare, minus eventuell rabatt.

SelfServiceCheckout - del 6

- a) Lag metoden **getMVA** i **Item**-klassen som beregner MVA som skal betales. Merk at prisen i **Item**-objektene inkluderer **allerede** MVA (slik som når du ser på en vare i butikken). Vi antar at MVA på alle varer er **25%**.
- b) Lag deretter metoden **getPriceWithoutMVA** i **Item**-klassen som returnerer prisen på denne varen **minus** MVA.

SelfServiceCheckout - del 7

a) Lag metoden **getTotalPriceForCart** som returnerer totalprisen på innholdet i handlekurven, **inkludert** eventuelle rabatter.

b) Lag deretter metoden **getTotalMVAForCart** som returnerer total MVA som skal betales for alle varer i handlekurven. Denne skal basere seg på sluttprisen fra oppgave a)

SelfServiceCheckout - del 8

SelfServiceCheckout må ha støtte for å skrive ut kvitteringer:

Hva	Pris	MVA	Total
1 x Tomato	4,25	0,75	5,00
2 x Cheese	25,50	4,50	60,00
1 x Burritos	38,25	6,75	45,00
Total MVA			16,50
Total			110,00
Takk for at du handlet hos oss i OOP!			

a) Lag en **toString**-metode som bruker de metodene vi har laget hittil for å få ut en fint formatert kvittering med riktige verdier. I akkurat denne oppgaven kan du se bort i fra at samme vare forekommer mer enn én gang i handlekurven.

(hvis tid) SelfServiceCheckout - del 9

Kvitteringen skriver nå ut riktig innhold i handlekurven, men slik som på vanlige kvitteringer så ønsker vi å gruppere like varer på kvitteringen.

- a) Ved å ta utgangspunkt i `#equals()`-metoden i **Item**-objektene, lag en hjelpemetode som teller antall identiske objekter i handlekurven og returnerer dette tallet.

- b) Legg til funksjonalitet i **toString**-metoden for å sjekke om en vare allerede er printet på kvitteringen. Hvis varen allerede er printet skal koden hoppe over denne varen. Bruk deretter metoden du lagde i a) til å printe ut riktige antall varer gruppert sammen på kvitteringen.

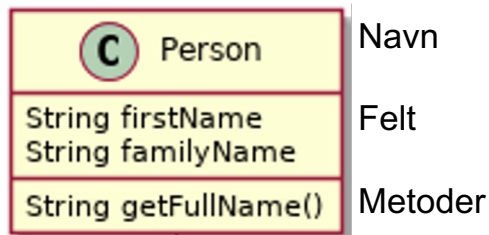
(Husk at hver vare vil nå printes ut kun én gang)

Hva	Pris	MVA	Total
1 x Tomato	4,25	0,75	5,00
2 x Cheese	25,50	4,50	60,00
1 x Burritos	30,25	0,75	45,00
Total MVA			16,50
Total			110,00
Takk for at du handlet hos oss i OOP!			

(hvis tid) Klassediagrammer

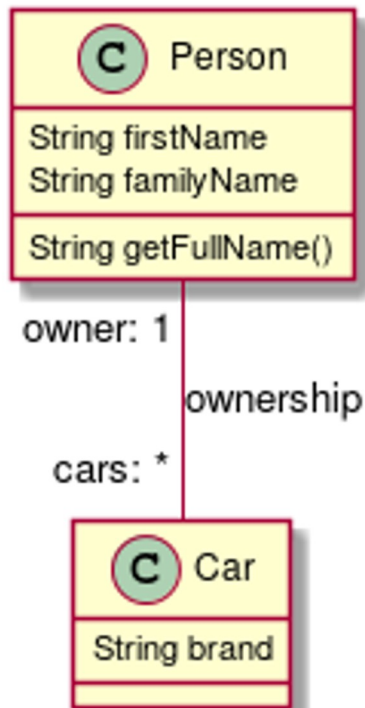
Repetisjon

- En del av pensum i TDT4100 er å kunne visualisere klasser/objekter/tilstander (m.m.).
- Vi visualiserer **klasser** med **klassediagram**:



(hvis tid) Fra kode til klassediagram

- Begge klasser har assosiasjon til hverandre i koden

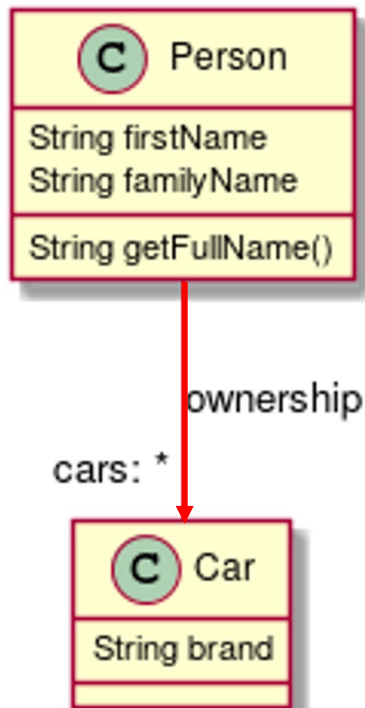


```
class Person {
    String firstName;
    String familyName;
    String getFullName();
    Collection<Car> cars;
}

class Car {
    String brand;
    Person owner;
}
```

(hvis tid) Fra kode til klassediagram

- Hvis en assosiasjon har et pilhode, så betyr det at den er enveis



```

class Person {
    String firstName;
    String familyName;
    String getFullName();
    Collection<Car> cars;
}

class Car {
    String brand;
}
  
```

Merk at her kan vi *navigere* fra **Person** til **Car**, men ikke fra **Car** til **Person**

(hvis tid) SelfServiceCheckout - del 10

Vi lager et forklarende **klassediagram** som viser oppbyggingen til klassene **SelfServiceCheckout** og **Item** og samspillet mellom disse.



Lykke til med ukas øving!

Spørsmål og tilbakemeldinger kan sendes til
eirik.l.tanberg@ntnu.no

