

Kræsjkurs med Tekna

Objektorientert Programmering, Java, TDT4100

Fredag 03.05.24 - 10:15 - F1

Kursholder Einar T. Myhrvold

github.com/einartmy01/TeknaKK



Tekna



Introduksjon

- Velkommen
- Oppklaring og formål for kræsjkurset
- Mentimeter:





Agenda

- Introduksjon
- Spesifikke tips
- Objekt? Klasse?
- Innkapsling
 - Eksempel
- Forvirrende modifikatorer
 - Eksempel
- Arv
 - Eksempel
- Interface
 - Eksempel
- Array og Lister
 - Eksempel

- Hashmap
 - Eksempel
- Delegering
 - Eksempel
- Observering
 - Eksempel - eksamen 2023
- Comparator, Comparable
- Iterator, Iterable



Ikke med: Lese/skrive fil & streams



Spesifikke tips øving

1. Programmering er ferskvare
 - 1.1. Lær dere konsepter og temaer nå, og få en skikkelig grind når det begynner å nærme seg.
2. Øv med en annen person!
 - 2.1. Å lese andres kode og dele sine løsninger er kjempenyttig.
3. Magnus Schjølberg

Fjern dere med co-pilot!



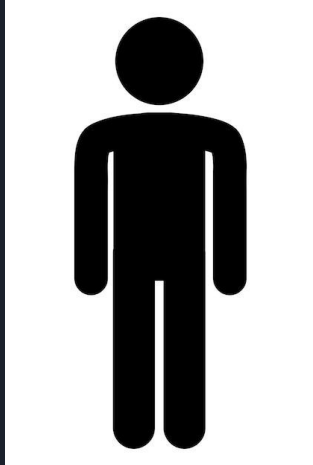
Spesifikke tips eksamen

1. Ikke bruk tid på å lete etter den mest fancy løsningen
 - 1.1. Hvis det funker så er det ofte godt nok
2. Fokuser på å få inn mest mulig av konseptene dere har lært på eksamen
 - 2.1. Hver oppgave er for å teste en spesifikk del av pensum
3. Svar på alle oppgaver uansett hva

Objekt? Klasse?

- En kneik som mange treffer på er forståelsen av hva en klasse og hva et objekt er for noe.

Klasse: Human



```
int age;  
  
double height;  
  
String name;
```

Objekt: human1



```
age = 24;  
  
height = 1.78;  
  
name = "Truls";
```





Innkapsling

- Hindre direkte tilgang til felt for andre klasser
- Dette gjøres med synlighetsmodifikatorer
 - **public, private, protected**
- Gir kun tilgang til det andre klasser trenger
 - get/set-metoder
- Bli vandt med innkapsling
 - Er riktig, og ser proft ut
 - Kan komme som “feilsøking” oppgave





De tre essensielle synlighetsmodifikatorene

public

→ Synlig for “alle”

protected

→ Synlig i klassen, i pakken og i subklasser

private

→ Synlig kun i klassen





Konstruktør

Konstruktør kommer etter variablene

Så og si alle klasser har en konstruktør

```
4  public class Human {  
5  
6      //Variabler som er forskjellig for vårt objekt av type menneske  
7      private int age;  
8      private double height;  
9      private String name;  
10  
11     // "Konstruktør", lage et nytt objekt av type menneske:  
12     public Human(int age, double height, String name){  
13         this.age = age;  
14         this.height = height;  
15         this.name = name;  
16     }
```





Eksempel av klasse/objekt, innkapsling



De litt mer forvirrende modifikatorene

final

- Ved å bruke denne så gjør man det umulig å senere endre verdien av en variabel
- Kan også brukes på metoder og klasser, men dette er ikke relevant i faget
- Merk: Selv om en liste er markert som **final** kan den fortsatt endres





De litt mer forvirrende modifikatorene

static

- Kan også brukes når vi har variabler som er felles for **alle** objekter i en klasse
- Variabelen/metoden tilhører hele **klassen**, og ikke en spesifikk instans.
- Kan brukes uten spesifikk objekt av klassen.





Eksempel av static og final med human





Arv

Hva:

- En metode for å la **subklasser** få egenskaper fra en **superklasse**
- Gjenbrukbar kode
- Speiler virkelige verden

Hvorfor:

- Klasser deler egenskaper
- Vi ønsker å gjenbruke kode
- Forenkler mange operasjoner
 - Lister med forskjellige typer elementer som kan ses på som en del av en overordnet “type”





Eksempel på arv - student



Grensesnitt / Interface

- Som klasser - men kan kun deklare variabler og metoder uten å definere noe innhold.
 - Merk: variabler deklarerert i grensesnitt blir *automatisk* **static final**
- Klasser som **implementerer** et grensesnitt må definere alle variablene og metodene i grensesnittet





Eksempel på interface - academic (og Proffessor)



Arrays

- Som lister i Python, men immutable
 - Dvs. kan ikke endre størrelse i etterkant
- Veldig primitiv i sammenligning med ArrayList
-
- Kan konvertere fra array til ArrayList på denne måten:

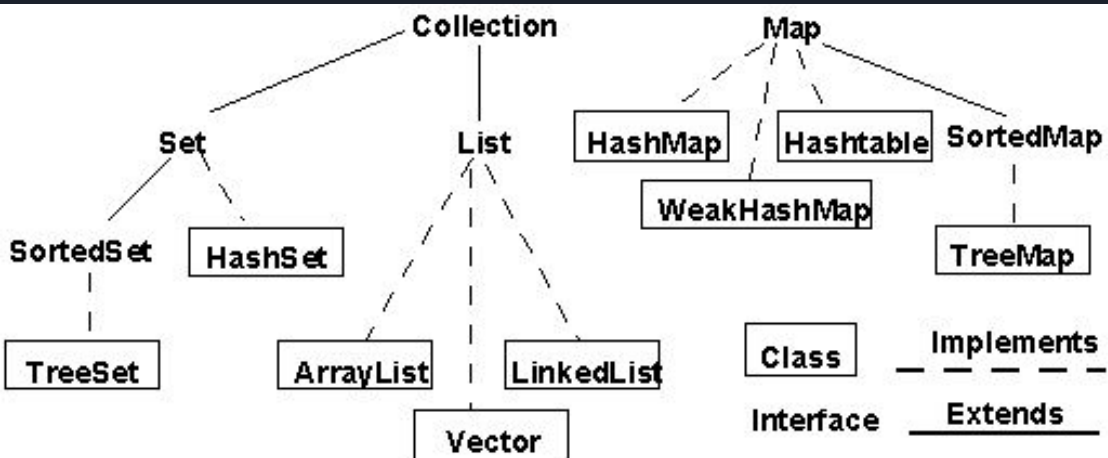
```
String[] array = {"Honda", "Ford", "BMW"};  
List<String> list = Arrays.asList(array);
```



Collections og Lister

Vi bruker

- Collection hvis vi ikke ønsker å aksessere på index
 - Har metodene
 - **add(element)**
 - **remove(element)**
 - **contains(element)**



- List hvis vi ønsker å aksessere på index
 - Legger til metodene
 - **get(index)**
 - **remove(index)**
 - **indexOf(element)**



question



Collections

```
Collection<Integer> list1 = new ArrayList<Integer>();  
List<Integer> list2 = new ArrayList<Integer>();
```

Hva er forskjellen på list1 og list2 i praksis?

- Dersom du trenger **indeks** -> Bruk List
- Hvis ikke -> Bruk Collection






ArrayList

- Kjenn til disse metodene:
 - **add(element)**
 - **remove(element)**
 - **contains(element)**
 - **get(index)**
 - **remove(index)**
 - **indexOf(element)**
 - **size()**
 - **isEmpty()**





Eksempel på arraylist - students in professor



HashMap

- Samme prinsipp som **dictionary** i Python
- Kan gjøre noe oppgaver enklere
 - Prioriter å bli rutinert på ArrayList

```
// Create a HashMap object called capitalCities
HashMap<String, String> capitalCities = new HashMap<String, String>();

// Add keys and values (Country, City)
capitalCities.put("England", "London");
capitalCities.put("Germany", "Berlin");
capitalCities.put("Norway", "Oslo");
capitalCities.put("USA", "Washington DC");
```





HashMap funksjoner

- Kjenn til disse metodene:
 - **put(key element, value element)** -> setter inn ny verdi
 -
 - **get(key element)** -> returnerer value til den “nøkkelen”
 -
 - **remove(key element)** -> fjerner nøkkel med verdi
 -
 - **containsKey(key element)** -> returnerer true hvis finnes





HashMap eksempel - Universitet med academic og subject



Delegering og observering

- 2 kapitler virkelig å hente inn poeng på.
- Ikke overkompliser for deg selv.





Delegering først

- En *programmeringsteknikk*, ikke en innebygd funksjon i Java
- Vi gir bare oppgaven videre til et annet objekt.
- Kommer ofte av 2 klasser som begge implementerer samme **Interface**.





Eksempel - endre academic, doAssignment



Observable - Observer

- Dette er også bare en teknikk, ikke noe som er innebygd i Java
- Teknikk for å kunne *lytte* på tilstand hos andre objekter.
- Navnene blir fort kronglete, men prøv å skille mellom grensesnitt og faktisk implementasjon
- Er ikke nødvendigvis sikkert at man *trenger* å implementere lytter-grensesnitt osv. men gjør det hvis du har tid.





Observering eksempel fra eksamen 2023

- Denne eksamen handlet om fly
- Egen oppgave om observering gikk ut på hvilken status flyet var i

Legg merke til:

- Når FlightStatus klassen har en endring i variablene sine. Så sender de informasjon om endringen ut.
 - Status endrer seg og sender ut den nye statusen
- FlightStatusObserverImpl motar denne informasjonen og gjør noe med det.
 - Får inn den nye statusen og printer så ut en melding om ny status.







Comparator og Comparable Interfaces

- Litt som observer og observable, men de er innebygde og eksisterer.
- Comparable<K> = Markerer klassen som at den kan sammenlignes med et objekt fra denne eller annen klasse
 - Funksjon: **CompareTo("Klasse" k)**
- Comparator<K> = Er en egen klasse som forteller om hvordan man sammenligner to elementer
 - Funksjon: **Compare("Klasse" k1, "Klasse" k2)**







Iterator og Iterable, Interfaces

- Iterable = Markerer klassen som at den kan itereres over på samme måte som en liste
- Iterator = Et objekt som lar oss iterere over en liste/collection.
 - Du bruker implisitt en iterator når du bruker en foreach-løkke
- Ofte har vi klasser som inneholder lister og som skal implementere Iterable-grensesnittet.







Hva burde man se etter???

Hvordan lese oppgavetekster



