Mer Java!

IN1010

Uke 2 (IN1010-tidsregning)

Tirsdag 27. januar 2026

Læringsmål

- Forrige uke
  - Grunnleggende trekk ved Java
  - Objekter og klasser i Java

- I dag: Bruk av Java API (klassebibliotekene)
  - Kunne lese og skrive til/ fra fil og terminal
  - Manipulere og konvertere strenger
  - Unntak (Exceptions)

  => Finne frem i dokumentasjon på klasser og metoder i Java API

- + litt til

Plan

- Hvordan bruke klasser i Java?

- Lese inn data fra fil i Java – med diverse tilbehør:
  - Java API (klassebiblioteket), packages, import og aksessmodifikatorer
  - Exceptions (unntakshåndtering)
  - Klassene File og Scanner
  - Lese ulike typer verdier

- Skrive til fil (PrintWriter).

- Tekststrenger (String), manipulering og konvertering. «Wrapper»-klasser.

- Kommandolinje-argumenter, beholdere og en ny for-løkke.

Liten repetisjon av **`static`**

- Ikke pensum i IN1000 – der brukte vi kun *instans*metoder og –variabler
- Statiske metoder og variabler tilhører klassen (ikke et bestemt objekt):
  - kan brukes selv om vi ikke har opprettet noen objekter
  - deles av alle objekter av klassen

- Kalles som `Klassenavn.xxx()`

- Eksempler:
  - `public static void main (…` // her starter utføring av alle Java programmer
// - det vil si før vi har opprettet noen objekter
  - `static int totalSum = 0;` // kan representere sum fra alle objekter
  - `static int teller = 0;` // kan representere antall objekter opprettet

Fra IN1000 uke 8: Python
Program med klasse i egen programfil

```
class Eksempel:
   def __init__(self):
        self._instvar1 = 0
```

eksempel.py

Klasser skrives ofte slik at de er nyttige i mange ulike programmer og kan gjenbrukes.

Eks: strenger og lister

```
from eksempel import Eksempel

def hovedprogram ():
    …
    # oppretter og bruker objekter

hovedprogram()
```

hovedprogram.py

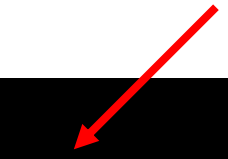
2 klasser i Java, den ene med hovedprogram

```
class Eksempel {
    int instVar = 0;
    // div annet innhold
}
```

Eksempel.java

Filnavn = Klassenavn + ".java"

Vi trenger ikke importere klasser som ligger i samme mappe.



```
class Hovedprogram {
    public static void main(String[] args) {
        // oppretter og bruker objekter
        Eksempel eks = new Eksempel();
    }
}
```

Hovedprogram.java

Kompilerer alt først
>javac *.java

Hovedprogrammet som utføres ligger i main-metoden – og kjøres her med kommandoen:
>java Hovedprogram

Lese fra fil

Se notatet: "Lesing og skriving i Java" på semestersiden

Navnestatistikk

- Hva er navnet på eldste person i en fil?

alder.txt

```
Petter 92
Kari 92
Emil 101
Katrine 99
```

FinnEldste. java

```java
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;


class FinnEldste {
    public static void main(String[] args) throws FileNotFoundException {
        File fil = new File("alder.txt");
        Scanner sc = new Scanner(fil);
        String eldsteNavn = "";
        int maksAlder = 0;
        while (sc.hasNextLine()) {
            String[] ord = sc.nextLine().split(" ");
            String navn = ord[0];
            int alder = Integer.parseInt(ord[1]);
            if (alder > maksAlder) {
                maksAlder = alder;
                eldsteNavn = navn;
            }
        }
        System.out.println("Eldste navn: " + eldsteNavn);
        System.out.println(eldsteNavn + "s alder: " + maksAlder);
        sc.close();
    }
}
```

Eller: try – catch, se notat

Kaller .close() når Scanner-objektet er en fil

Bli kjent med Java API

- Java biblioteket (Java API) inneholder alt dere trenger og MYE mer

  => Bli vant til å bruke Java API dokumentasjon allerede nå

- Java API dokumentasjon på nett:

  https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html

  Java 8 inneholder alt vi bruker i IN1010, lettere å finne frem i!

- Big Java: Mer begynnervennlige forklaringer av det meste du trenger

Hvorfor API og lærebok i stedet for KI?

Ikke enten eller:

- KI-tjenester kan være nyttige for mer sammensatte problemer, når du ikke vet hvor du skal starte og du trenger oversikt du ikke har (og ikke vil ha?)

Men

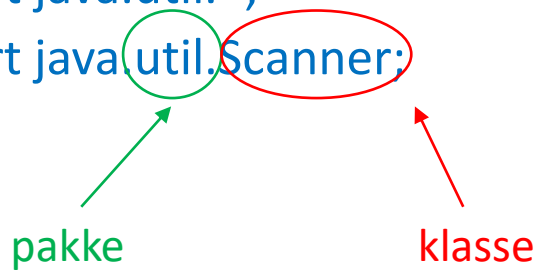
- å kunne slå opp i dokumentasjon er en nødvendig ferdighet
- Informasjonen du får er korrekt
- Du kan få en mer helhetlig forståelse/ oversikt over det du slår opp
- Du risikerer å lære noe du ikke visste du trengte
- Du må kanskje jobbe litt mer med å koble det du jobber med og informasjonen du trenger => tankearbeid er BRA

Pakker og klasser, import

=> trenger vanligvis ikke tenke på dette

- All Java-kode hører til en klasse, som igjen hører til i en pakke (package)
  - Klasser uten pakke legges i en *unnamed* pakke
  - Disse er tilgjengelige for programmer i samme mappe (hvis filnavn = klassenavn)
- Pakken java.lang importeres alltid automatisk (her ligger for eksempel String-klassen)

- Andre pakker (eller enkeltklasser i pakker) må importeres eksplisitt
  - import java.util.*;
  - import java.util.Scanner;



Hvor ligger Scanner? Søk "java api 8 Scanner"

Tilgangskontroll: *Hvorfra* kan dette aksesseres?

Tilgangskontroll

- Alle klasser i programmet ditt ligger i samme mappe => de er i samme pakke

- Om du ikke angir tilgangsnivå er alt *package-private* (default tilgang)
  - Alle dine programmer i denne mappen har full tilgang til alt
  - Men: **main** må være **public** (fordi kjøresystemet til Java skal finne den)

- Innhold i klassen som du ønsker å beskytte (data og hjelpemetoder)
  - angis som **private** (kan bare aksesseres i klassen)

Innkapsling, «black box»

- Mer å ta hensyn til og flere muligheter om programmene dine skal brukes av andre, eller gjenbrukes senere av deg selv – og når vi begynner å bruke arv

Feilhåndtering – "unntak"

- Feil som oppstår under kjøring genererer unntak (exceptions) i Java (og i Python!)

- Noen unntak *krever* Java at vi håndterer i programmene våre – blant annet ved åpning av filer
  - krever import av det aktuelle unntaket
  - unntak kan fanges og håndteres der det oppstår (med **`try..catch`**)
  - .. eller "kastes" (med **`throws`**) fra metoden tilbake til kallstedet
  - I siste tilfelle må det håndteres i kall-metoden (med **`throws`** eller **`try..catch`**)
  - Flere eksempler og detaljer senere i semesteret

- Eksempler på **`try..catch`** i notatet "Enkel lesing og skriving i Java" på semestersiden.

- "Alt" om unntakshåndtering i Big Java, 7.4.

Håndtering av unntak med "throws"

Eksempel fra FinnEldste-programmet

```
.
.
class FinnEldste {
    public static void main(String[] args) throws FileNotFoundException {
        File fil = new File("alder.txt");
.
.
```

- Om det er greit at programmet stopper med en generell melding
- Må "kastes" av hver metode opp til main

Håndtering av unntak med try .. catch

Eksempel fra Notatet: Lese og skrive i Java

```
class LagFil {
    public static void main(String[] args) {
        PrintWriter f = null;
        try {
            f = new PrintWriter("tall-til-10.txt");
        } catch (Exception e) {
            System.out.println("Kan ikke lage filen tall-til-10.txt.");
            System.exit(1);
        }
```

- Kan få en tilpasset melding/ håndtering
- Trenger ikke kjenne til/ håndtere i andre metoder

FinnEldste. java

```java
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;


class FinnEldste {
    public static void main(String[] args) throws FileNotFoundException {
        File fil = new File("alder.txt");
        Scanner sc = new Scanner(fil);
        String eldsteNavn = "";
        int maksAlder = 0;
        while (sc.hasNextLine()) {
            String[] ord = sc.nextLine().split(" ");
            String navn = ord[0];
            int alder = Integer.parseInt(ord[1]);
            if (alder > maksAlder) {
                maksAlder = alder;
                eldsteNavn = navn;
            }
        }
        System.out.println("Eldste navn: " + eldsteNavn);
        System.out.println(eldsteNavn + "s alder: " + maksAlder);
        sc.close();
    }
}
```

Eller: try – catch, se notat

Kaller .close() når Scanner-objektet er en fil

Klassen String

(og litt om wrapper-klasser / innpakningsklasser)

"Wrapper" klasser for primitive typer

Integer, Float, Boolean, Character, ++

Mer overhead, mer funksjonalitet om verdiene ligger i objekter med metoder:

+ Noen ganger trenger vi at en verdi er et objekt (eksempler i Beholder-forelesningene)
+ Disse klassene kan brukes som verktøykasser - med statiske metoder og konstanter som kan brukes uten å opprette et objekt av klassen. Eksempler fra **class Integer**:

  - Metoden **`parseInt`**

```
int alder = Integer.parseInt(biter[1]);      // konverterer String til int
```

  - Konstanten **`MAX_VALUE`**

```
int max = Integer.MAX_VALUE;                 //  høyeste verdi for integer
```

Konvertering fra String

- Klassene Integer, Double, Boolean, Char, .. har metoder for å hente verdier av sine respektive typer fra en String

```
String minStreng = " 1243 ";
minStreng = minStreng.trim();
System.out.println(Integer.parseInt(minStreng));


minStreng = "3.14";
System.out.println(Double.parseDouble(minStreng));


minStreng = "false";
System.out.println(Boolean.parseBoolean(minStreng));
```

- NB: argumentet må *kun* inneholde verdien som skal konverteres. Whitespace kan fjernes med String-metoden trim()

Konvertering til String

- Klassene Integer, Double, Boolean, Char, .. har alle en metode toString som tar et argument av respektiv type og returnerer den som en String

```
int tall = 1234;
String minStreng = Integer.toString(tall);

double flyttall = 12.34;
String minStreng2 = Double.toString(flyttall);

System.out.println("minStreng = " + minStreng);
System.out.println("minStreng2 = " + minStreng2);
```

- Merk at vi **her** ikke bruker wrapper-klassene til å representere verdiene – vi utnytter bare de statiske metodene i klassene til å konvertere primitive typer

Konvertering til String på ulike måter

- Legge til en tom streng

```
double tall = 5.2;
String s = "" + tall;
```

- Bruke Double klassens (statiske) metode toString

```
double tall = 5.2;
String s = Double.toString(tall);
```

Bearbeiding av tekststrenger

String er immutable som i Python: *Ny streng returneres*

- Klassen String inneholder mange nyttige metoder
  - `s.charAt(pos)` returner karakteren på posisjon pos i s
  - `s.equals(s2)` returnerer true hvis s og s2 er like tegn for tegn (ikke bruk s==s2)
  - `s.substring(3,5)` returnerer kopi av innhold i posisjon 3-4 som ny streng
  - `split`
  - `toLowerCase` og `toUpperCase`
  - `trim`

- Du finner alt om String i Java API (og et nyttig utdrag i Big Java)

- Java metoder kan "overloades" => finnes i flere varianter:
  - Samme navn og type, men ulike parametere.
  - Den metode som passer med argumentene i kallet, er den som blir utført.

Klassen Scanner

"scanner" er et buffer – nyttig for lesing fra fil, fra terminal eller fra en String

Klassen Scanner

- **Scanner** ligger i pakken **java.util**, som må importeres

- Konstruktøren til Scanner tar et argument som angir hvor du skal lese fra
  - `System.in` (terminalen, referanse til et **InputStream**-objekt) eller
  - referanse til et objekt av klassen **File** (om du skal lese fra fil) eller
  - referanse til et **String**-objekt (om du skal lese fra en streng))

- Scanner-klassen tilbyr to hovedmåter å lese inn data på:
  - en hel linje (som så skal lagres eller prosesseres videre)
  - ett og ett "token" (på norsk "ord")

En linje av gangen som tekst

- I prinsippet det vi gjorde i Python i IN1000: Leser en og en linje inn i en String – trenger ikke vite noe om innholdet på linjen på forhånd

- Bruker Scanner-metoder for å sjekke og deretter lese, en og en linje:
  - `public boolean hasNextLine()`
  - `public String nextLine()`

- Leser forbi linjeskift, returner alt *før* linjeskift som en String*

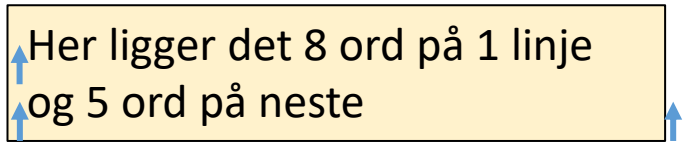
Ulikt Python
(som tok med linjeskift
når vi itererte over
linjene i en fil)

- Kan siden bruke linjen hel, eller dele opp (som i eksempelet)

* "Kaster" linjeskiftet

Scanner har en posisjonspeker

- Jobber seg gjennom et buffer (innhold fra terminalen, en fil eller en String)
- Har en *posisjonspeker* (pos) som holder rede på hva som er lest hittil



- `nextLine` *leser* alltid fremover fra pos (og flytter da pos)

```
s = nextLine();       // s == "Her ligger det 8 ord på 1 linje"
t = nextLine();       // t == "og 5 ord på neste"
```

- `hasNextLine` *sjekker* hva som kommer fra pos - *uten* å flytte pos

```
mer = hasNextLine()   // returnerer true eller false
                      // nå er mer == false
```

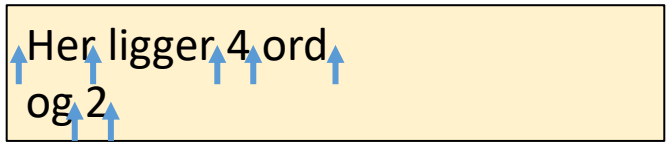
Å lese ett og ett "token" ("ord")

- Et token er en sammenhengende tegnsekvens som avsluttes med whitespace* eller slutt på filen
- Kalles gjerne *ord* på norsk (kan være tall eller andre tegn)

- Viktige metoder for å behandle ord i klassen Scanner :

```
public boolean hasNext() // er det noen flere ord?
public String next()     // les og returner neste ord som en String
```

- `next` og `hasNext` hopper over *innledende* whitespace (men `hasNext` flytter ikke pos).



* whitespace: blank, tab, linjeskift

Å lese andre typer enn String

- Kan bruke Scanner-metoder for å teste neste ord:
  - `public boolean hasNextInt()` (sjekk om neste ord er int)
  - `public boolean hasNextDouble()` (sjekk om neste ord er double)

- Kan deretter lese med riktig metode, f eks:
  - `public int nextInt()` (les og returner neste ord som en int)
  - `public double nextDouble()` (les og returner neste ord som en double)

Lese ett og ett ord fra fil

- Vi bruker **Scanner** som før
  - Må først opprette et objekt av klassen **File** (ligger i **java.io**)
  - Dette sendes som argument til nytt **Scanner**-objekt

```
class LesFraFilTokens {
    public static void main(String[] attr)
            throws FileNotFoundException {
        File minFil = new File("Handleliste.txt");
        Scanner lesFil = new Scanner(minFil);
        while (lesFil.hasNext()) {
            String vare = lesFil.next();
            System.out.println(vare);
        }
    }
}
```

Handleliste.txt

```
bok laptop pizza
  mobil hus
bil

glass kaffe
```

Trening i fillesing og Scanner:
Skriv finnEldsteNavn med innlesing ord for ord

Test av Scanner metoder – hva skjer?

`\n` angir linjeskift

```
import java.util.Scanner;


class LesTokens {
    public static void main(String[] args) {
        String test = "    1kjdfs\n834756 2.3";
        Scanner les = new Scanner(test);
        System.out.println(les.next());
        System.out.println(les.nextInt());
        System.out.println(les.hasNextInt());
        System.out.println(les.nextDouble());
    }
}
```

Utskrift:

1kjdfs

834756

false

2.3

Skrive til fil

Se notatet: "Lesing og skriving i Java" på semestersiden

Å skrive til fil

```java
import java.io.PrintWriter;
import java.io.FileNotFoundException;

class SkrivTilFil{
    public static void main (String[] args) throws FileNotFoundException {
        PrintWriter utfil = new PrintWriter("utfil.txt");

        utfil.println("Linje 1");
        utfil.close();
    }
}
```

Lese fra terminal

Se notatet: "Lesing og skriving i Java" på semestersiden

Navn og alder (I)

Mest oversiktlig:
Bruk nextLine()

```
import java.util.*;
class LesFraTermNavnAlder {
    public static void main (String [ ] args) {
        int alder;
        String navn;
        Scanner minInn = new Scanner (System.in);
        System.out.print(" Skriv navn: ");
        navn = minInn.nextLine();   // nextLine leser forbi linjeskift
        System.out.print(" Skriv alder: ");
        alder = minInn.nextInt();   // nextInt leser fra starten av neste linje

        System.out.println(" Du heter " + navn + " og er " + alder + " aar.");
    }
}
```

Alder og navn (II)

Mest oversiktlig:
Bruk nextLine()

```
import java.util.*;
class LesAlderNavn {
    public static void main (String [ ] args) {
        int alder;
        String navn;
        Scanner minInn = new Scanner (System.in);

        System.out.print(" Skriv alder: ");
        alder = minInn.nextInt();   // nextInt leser ikke forbi linjeskift
        System.out.print(" Skriv navn: ");
        navn = minInn.nextLine();   // nextLine leser forbi 1. linjeskift
        // dvs stopper etter det linjeskiftet som "lå igjen" etter nextInt.
        // I variabelen navn legges det dermed bare en tom streng

        System.out.println(" Du heter " + navn + " og er " + alder + " aar.");
    }
}
```

Et lite summeringsprogram

Argumenter på kommando-linjen

For-løkke for elementer i en beholder

Argumenter fra kommandolinjen

```
PS M:\Ifi\Undervisning\IN1010 V2026\Forelesninger\Uke 2\Kode>
PS M:\Ifi\Undervisning\IN1010 V2026\Forelesninger\Uke 2\Kode> java SummerDisse 3 2 1
Summen er 6
PS M:\Ifi\Undervisning\IN1010 V2026\Forelesninger\Uke 2\Kode>
```

```java
class SummerDisse {
    public static void main(String[] args) {

        int sum = 0;
        for (int i = 0; i < args.length; i++) {
            sum += Integer.parseInt(args[i]);
        }

        System.out.println("Summen er " + sum);
    }
}
```

Argumenter fra kommandolinjen

- Merk bruken av argumenter fra kommandolinjen – disse mottar vi som en String array
- Lengden av arrayen finner vi med .length
  (NB: Ikke et metodekall for arrayer!)
- Arrayen args er av typen String => Må konvertere hver streng

- Kan brukes også for enkeltverdier – for eksempel et filnavn

```
import java.io.PrintWriter;

class LagFil {
    public static void main(String[] args) {
        PrintWriter f = null;
        try {
            f = new PrintWriter(args[0]);
        } catch (Exception e) {
            System.out.println("Kan ikke lage filen " + args[0]);
            System.exit(1);
        }
        // Skriv tallene fra 1 til 10:
        for (int i = 1; i <= 10; i++) {
            f.print(" " + i);
        }
        f.println(); f.close();
    }
}
```

Forenklet/ for-each/ enhanced for-løkke

- Ligner for-løkken i Python
- Går gjennom alle verdier i en beholder (her en array)

```
class SummerDisseFor {
    public static void main(String[] args) {

        int sum = 0;
        for (String str: args) {
            sum += Integer.parseInt(str);
        }
        System.out.println("Summen er " + sum);
    }
}
```

- Bruker "vanlig" for-løkke
  - Hvis vi trenger indeks-verdien inne i løkken
  - Hvis vi skal endre verdier i arrayen

ArrayList: En beholder-klasse for objekter

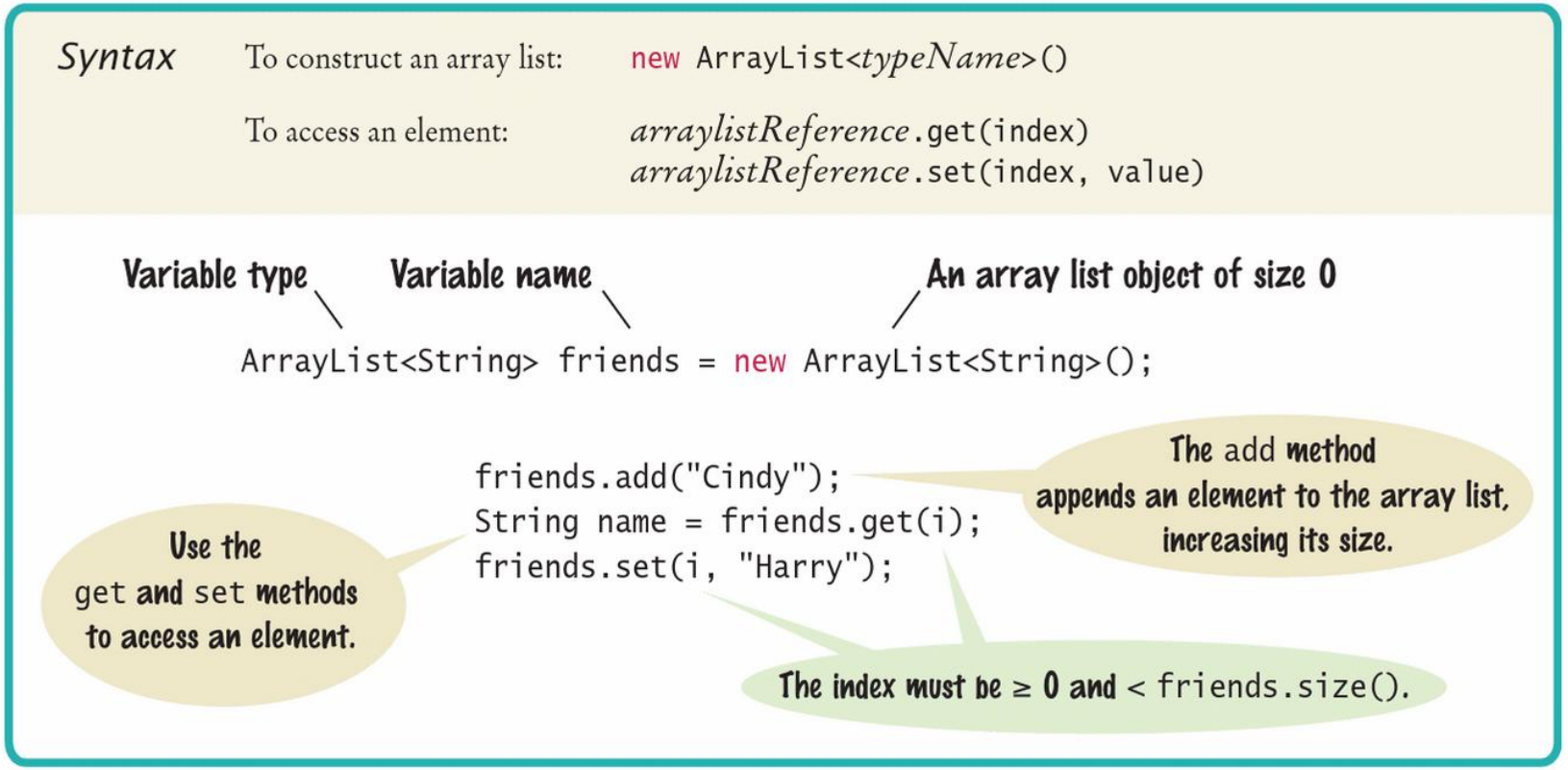
- Vi kan kalle metoder på ArrayList-objektene (i motsetning til arrayer)
- Fleksibel størrelse, mer funksjonalitet, tyngre å kjøre og mer tungvint å bruke
- Ligner på Python List funksjonalitet

- Må oppgi hvilken type elementene skal ha (alle må ha samme type)
- Kun for (referanser til) objekter, ikke primitive verdier (som `int`), men wrapper-klasser er ok

Bruk

- Importer Java.util.ArrayList
- Deklarerer en peker til ArrayList for riktig type elementer, oppretter ArrayList objekt
- Kan deretter kalle metoder på dette ArrayList-objektet

ArrayList: En beholder-klasse for objekter



- Se Big Java kapittel 6.8 og Java API om ArrayList

Mer om arrayer, ArrayList, for-løkker, ...

Big Java – elektronisk tilgjengelig fra semestersiden under Pensum

- Forklaringer, tegninger, detaljer
- ... eksempler, tips inkludert "mønstere" for bruk

- Arrayer kapittel 6
  - Forenklet for-løkke i 6.2
- Løkker kapittel 4

Oppsummering

Bruk teorien **mens du programmerer** – og **programmer** mens du leser

- Slå opp klasser og metoder i Java 8 API
- Bruk lærebok eller Java tutorials (Java 8) for mer detaljerte forklaringer
- Oppskrifter på lese fra og skrive til fil og terminal i notat på semestersiden

- Exceptions gir nyttig og noen ganger nødvendig håndtering av feilsituasjoner

- Primitive typer kan pakkes inn i objekter av tilsvarende klasse Integer, Double, Boolean, .. Disse "wrapper"-klassene har også nyttige statiske metoder (og konstanter)
- Bli kjent med String-klassen for manipulasjon og konvertering
- Array er en effektiv, nyttig og veldig vanlig konstruksjon i mange språk
- Java har to ulike former for for-løkker