Java for deg som kan Python (Del 2)

Leif Harald Karlsen

Universitetet i Oslo

16.01.24



Teknologi, språkmodeller og læring

- Mye ny og gammel teknologi som kan løse problemer
- Store språkmodeller (som ChatGPT, Bing, osv.) er flotte verktøy!
- Vil nok klare mange av oppgavene i dette kurset
- Derfor to viktige spørsmål:
  - Er det noen grunn til å lære seg pensum?
  - Hvordan bruke store språkmodeller i studiene?

Er det noen grunn til å lære seg pensum? (1)

JA! :D

- Språkmodeller er langt flinkere på problemene i dette kurset enn pensum generelt!
  - Problemene dere får er “enkle” og vanlige
  - Modellene er derfor godt trent på disse, men ikke “ekte” problemer
  - Modellene sliter med å sette sammen mindre program-snutter til større løsninger
  - Vi må lære oss det enkle før vi kan forstå det vanskelige
- Ikke alltid mulig å bruke språkmodeller
  - Sensitive domener
  - For store eller komplekse problemer
  - Dette kan endre seg, men vanskelig å spå fremtiden

Er det noen grunn til å lære seg pensum? (2)

JA! :D

- Må gi riktig input/prompt
  - Krever ofte dyp forståelse av domenet
  - Må vite hva som er mulig og ikke
- Språkmodellene gjør feil
  - Feil i selve løsningen
  - Løser feil problem
  - Svarene må derfor sjekkes
- Språkmodellene kan kun det den kan
  - Nye teknologier kommer og går
  - Kan ikke forskning og utvikling
- Det er gøy!

Hvordan bruke store språkmodeller i studiene?

- Ikke bruk språkmodeller til å løse oppgaver!
  - Læring krever at man faktisk tenker
  - Lærer lite av å bare se en løsning
- Bruk dem heller for samtaler om pensum!
  - Still generelle spørsmål og oppfølgingsspørsmål
  - Husk: De gjør fortsatt feil! Test ut svaret den gir
- Læring er noe du må gjøre selv og kan ikke automatiseres!

Klasser

- I Java må all kode ligger inni en klasse
- Klassene har samme formål som i Python
  - Pakker sammen data og funksjonalitet på dataene
  - Dataene er lagret i felter/variable inni klassen
  - Funksjonaliteten er metoder inni klassen
- En klasse definerer et mønster for objekter
- Kan lage et objekt fra en klasse med nøkkelordet `new`
  - F.eks. `Person p = new Person();`
- Feltene kommer øverst i klassen, og metodene deretter
- Java har derimot mye mer funksjonalitet rundt klassene sine

```
class Person {

    // Felter

    // Metoder
}
```

Felter

- Er bare variabler (som vi så forrige uke) men definert rett i klassen
- Må ha type!
- Kan være pekere til andre objekter av andre klasser
- Kan hente ut et felt fra et objekt med `.`
  - F.eks. `p.navn` eller `p.alder`

```
class Person {

  String navn;
  int alder;

  ...
}
```

```
// Eksempel bruk:
Person p = new Person();
p.navn = "Kari";
p.alder = 22;

System.out.println(p.navn + " er "
         + p.alder + " år gammel.");

// -> "Kari er 22 år gammel."
```

Metoder

- Metoder implementerer oppførsel til klassen
- Metodene kan bruke klassens variabler
- Metoder defineres på formen

  `<type> <navn>(<parametere>) { <kode> }`
- Parameterene er variabel-deklarasjoner med type og navn
- Typen før navnet angir typen på returverdiene
- Dersom metoden ikke returnerer noe er returntypen `void`
- For å skille mellom globale variable og klassens variable kan man bruke `this`
  - Tilsvarende Pythons `self`

```
class Person {

  String navn;
  int alder;

  String hentNavn() {
    return this.navn;
  }

  int hentAlder() {
    return this.alder;
  }

  void byttNavn(String nyttNavn) {
    this.navn = nyttNavn;
  }
}
```

Metoder: Bruk

```java
class Person {

  String navn;
  int alder;

  String hentNavn() {
    return this.navn;
  }

  int hentAlder() {
    return this.alder;
  }

  void byttNavn(String nyttNavn) {
    this.navn = nyttNavn;
  }
}
```

```java
// Eksempel bruk:
Person p = new Person();
p.navn = "Kari";
p.alder = 22;

System.out.println(p.hentNavn() + " er "
          + p.hentAlder() + " år gammel.");

// -> "Kari er 22 år gammel."

p.byttNavn("Kari Olsen");

System.out.println(p.hentNavn() + " er "
          + p.hentAlder() + " år gammel.");

// -> "Kari Olsen er 22 år gammel."
```

Konstruktør

- Spesiell metode som er ansvarlig for å initialisere variablene i klassen
- Tilsvarer Pythons `__init__`
- Har ingen returtype og samme navn som klassen
- Kan ha flere konstruktører, så lenge de har ulike argumenter
  - Kan alltid ha flere metoder med samme navn, så lenge de har ulike argumenter
  - Enten ulikt antall argumenter, eller argumentene har ulik type

```
class Person {

  String navn;
  int alder;

  Person() {
    this.navn = "Ukjent";
    this.alder = 0;
  }

  Person(String navn, int alder) {
    this.navn = navn;
    this.alder = alder;
  }

  ...
}
```

Konstruktør: Bruk

```
class Person {

  String navn;
  int alder;

  Person() {
    this.navn = "Ukjent";
    this.alder = 0;
  }

  Person(String navn, int alder) {
    this.navn = navn;
    this.alder = alder;
  }

  ...
}
```

```
// Eksempel bruk:
Person p1 = new Person();

System.out.println(p1.hentNavn() + " er "
          + p1.hentAlder() + " år gammel.");

// -> "Ukjent er 0 år gammel."

Person p2 = new Person("Kari Olsen", 22);

System.out.println(p2.hentNavn() + " er "
          + p2.hentAlder() + " år gammel.");

// -> "Kari Olsen er 22 år gammel."
```

Objekter

- Viktig å skille mellom følende 3 ting:
  1. en *klasse*
  2. et *objekt* av en klasse
  3. en variabel som peker på et objekt, ofte kalt en *referanse*
- I eksempelet er `Person` en klasse og `p1`, `p2` og `p3` referanser til `Person`-objekter
- Merk: `p2` og `p3` er to forskjellige *referanser* som peker på samme objekt
- (Mer om dette neste uke)

```
Person p1 = new Person();

Person p2 = new Person("Kari", 22);

Person p3 = p2;

p3.byttNavn("Kari Olsen");

System.out.println(p2.hentNavn());

// -> "Kari Olsen"
```

Eksempel: Teller

Python:

```python
class Teller:


    def __init(self)__:
        self.antall = 0


    def hent_tall(self):
        return self.antall


    def tell(self):
        self.antall += 1


    def tellOpp(self, n):
        self.antall += n
```

Java:

```java
class Teller {

  int antall;

  Teller() {
    this.antall = 0;
  }

  int hentAntall() {
    return this.antall;
  }

  void tell() {
    this.antall++;
  }

  void tellOpp(int n) {
    this.antall += n;
  }
}
```

Eksempel: Teller-kjøring

```java
class Teller {

  int antall;

  Teller() {
    this.antall = 0;
  }

  int hentAntall() {
    return this.antall;
  }

  void tell() {
    this.antall++;
  }

  void tellOpp(int n) {
    this.antall += n;
  }
}
```

```java
// Vanlig å lage en egen klasse for kjøring
// som da er inngangen til resten av programmet
class TellerKjorer {
  public static void main(String[] args) {

    Teller t1 = new Teller();
    t1.tell();

    Teller t2 = new Teller();
    t2.tell();
    t2.tellOpp(2);

    System.out.println("Antall t1: " + t1.hentAntall());
    // -> Antall t1: 1

    System.out.println("Antall t2: " + t2.hentAntall());
    // -> Antall t2: 3
  }
}
```

Static

- `static` er et nøkkelord som kan brukes på både felter og metoder
- Kalles *statiske felter* og *statiske metoder* på norsk
- Et statisk felt er et felt som kun finnes én gang
  - Altså ikke én gang for hvert objekt, slik som vanlige felter
  - Feltet er altså felles for alle objektene av klassen
  - Kan også tenkes på som en klasse-variabel
  - Kan brukes uten å ha et objekt, f.eks. `Person.MYNDIG_ALDER`
- Tilsvarende, en statisk metode en metode som er felles for alle objekter
  - En statisk metode kan derfor ikke bruke `this`
  - Statiske metoder kan derfor kun bruke statiske felter eller globale variable
  - Kan også kalles direkte fra klassen, f.eks. `Person.erMyndigAlder(19)`
- `main`-metoden er alltid statisk siden den skal kalles uten et objekt

Eksempel: Statisk konstant

Lag en variabel som sier når en person er myndig:

```java
class Person {

  static int MYNDIG_ALDER = 18;
  String navn;
  int alder;

  ...

  boolean erMyndig() {
    return MYNDIG_ALDER <= this.alder;
  }

  static boolean erMyndigAlder(int alder) {
    return MYNDIG_ALDER <= alder;
  }
}
```

```java
Person p = new Person("Kari", 22);

System.out.println("Kari er myndig: "
                   + p.erMyndig());
// -> "Kari er myndig: true"

System.out.println("17 er myndig alder: "
                   + Person.erMyndigAlder(17));
// -> "17 er en myndig alder: false"
```

Eksempel: Teller 3

Lag en variabel som holder på totalen for alle tellere:

```
class Teller {
  int antall;
  static int totalt = 0; //statisk variabel

  Teller() {
    this.antall = 0;
  }

  int hentAntall() {
    return this.antall;
  }

  void tell() {
    this.antall++;
    totalt++;             // ikke this foran statiske
  }

  static int hentTotal() {
    return totalt;
  }
}
```

Private og public

- `private` gjør at feltet/metoden kun kan brukes av klassen den er i
- `public` gjør at feltet/metoden kan brukes av alle
- `protected` gjør at feltet/metoden kun kan brukes av klasser i samme mappe/pakke
  - Denne er standard (brukes om man ikke skriver noe)
- Disse brukes for å gjemme detaljer
- Gjøre det tydelig hva som skal og ikke skal brukes av andre klasser
- Gir programmereren kontroll og kan gjøre antagelser om koden

Eksempel: Teller 4

```
public class Teller {              // alle kan bruke klassen

  private int antall;              // kan nå ikke nås utenfor klassen
  private static int totalt = 0;   // kan nå ikke nås utenfor klassen

  public Teller() {
    this.antall = 0;
  }

  public int hentAntall() {        // kan kalles utenfor klassen
    return this.antall;
  }

  public void tell() {
    this.antall++;
    totalt++;
  }

  public static int hentTotal() { // kan kalles utenfor klassen
    return totalt;
  }
}
```

Array

- Array: slags liste med verdier med lik type
- Arrayer har type lik typen til verdiene etterfulgt av `[]`
- Et array har en konstant størrelse som oppgis når det lages
- Lages ved å bruke `new`
- Kan også lages direkte ved å liste opp verdier mellom krøllparenteser
- Kan hente ut eller tilordne verdier i arrayet basert på deres plassering i arrayet
- Plasseringen blir ofte kalt *indeksen*
- Lengden til et array finnes i `length`
- Arrayer er veldig effektive å bruke, men har konstant størrelse

```
String[] s = new String[3];
int[] tall = {2, 4, 8};

s[0] = "Hei";
s[1] = "alle"; //s[2] er null

System.out.println(s[0] + " " + s[1]);
// -> "Hei alle"

for (int i = 0; i < tall.length; i++) {
    System.out.println(tall[i]);
}
// -> 2
// -> 4
// -> 8

System.out.println(tall[3]);
// -> ERROR
s[3] = "hadet";
// -> ERROR
```

Litt større eksempel: Kalender

Vi skal lage et program som:

- holder styr på kalender-hendelser
- en hendelse skal kunne ha en dato, et klokkeslett og en beskrivelse
- hvis ingen dato er oppgitt skjer hendelsen daglig
- ønsker å kunne skrive ut alle dagens hendelser

Litt større eksempel: Hendelse-implementasjon

```java
public class Hendelse {
  private String dato;
  private String klokkeslett;
  private String beskrivelse;

  public Hendelse(String dato, String klokkeslett, String beskrivelse) {
    this.dato = dato;
    this.klokkeslett = klokkeslett;
    this.beskrivelse = beskrivelse;
  }

  public Hendelse(String klokkeslett, String beskrivelse) {
    this(null, klokkeslett, beskrivelse); // Kaller metoden over
  }

  public boolean iDag(String idag) {
    return this.dato == null || this.dato.equals(idag); // "||" lik Pythons "or"
  }

  public void skrivUt() {
    System.out.println(this.beskrivelse + " (" + this.klokkeslett + ")");
  }
}
```

Litt større eksempel: Kalender-implementasjon

```
public class Kalender {
  private static int STANDARD_STORRELSE = 50;
  private Hendelse[] hendelser;
  private String navn;

  public Kalender(String navn, int antall) {
    this.hendelser = new Hendelse[antall];
    this.navn = navn;
  }
  public Kalender(String navn) {
      this(navn, STANDARD_STORRELSE);
  }

  public String hentNavn() {
    return this.navn;
  }

  public void settInnHendelse(Hendelse nyHendelse) {
    int i = 0;
    while (this.hendelser[i] != null) { i++; }
    this.hendelser[i] = nyHendelse;
  }

  public void skrivUtDagensHendelser(String idag) {
    for (Hendelse hendelse : this.hendelser) {
      if (hendelse != null && hendelse.iDag(idag)) { // "&&" lik Pythons "and"
        hendelse.skrivUt();
      }
    }
  }
}
```

Kalender-bruk

```
public class KalenderMain {
  public static void main(String[] args) {
    Kalender studiekalender = new Kalender("Studier", 20);
    Kalender fritidskalender = new Kalender("Hobby");

    Hendelse f1 = new Hendelse("2024-01-16", "14:15", "Forelesning 2 i IN1010");
    Hendelse f2 = new Hendelse("2024-01-22", "10:15", "Forelesning 3 i IN1010");

    studiekalender.settInnHendelse(f1);
    studiekalender.settInnHendelse(f2);

    Hendelse fest = new Hendelse("2024-01-16", "19:00", "Fest");
    Hendelse middag = new Hendelse("17:00", "Middag");
    fritidskalender.settInnHendelse(fest);
    fritidskalender.settInnHendelse(middag);

    System.out.println("Studier i dag:");
    studiekalender.skrivUtDagensHendelser("2024-01-16");
    System.out.println(); // blank linje
    System.out.println("Fritid i dag:");
    fritidskalender.skrivUtDagensHendelser("2024-01-16");
  }
}
```

Større eksempel: Forbedringer

(Anta vi har en litt mer fancy `KalenderMain`-klasse, slik at dette blir et kjørbart program, skal fikse dette neste uke)

Etter at programmet har blitt brukt av ulike folk, har de følgende forslag:

- Bør slippe å måtte gi inn dagens dato i en kalender-app!

Forbedringer – Implementasjon

```java
import java.time.LocalDate; // Importerer en klasse fra Javas innebygde bibliotek

public class Hendelse {

  private LocalDate dato;  // Bruker den fremfor enkel String
  private String klokkeslett;
  private String beskrivelse;

  public Hendelse(String dato, String klokkeslett, String beskrivelse) {
    if (dato != null) {  // Parser dato-strengen til en LocalDate
      this.dato = LocalDate.parse(dato);
    }
    this.klokkeslett = klokkeslett;
    this.beskrivelse = beskrivelse;
  }

  public Hendelse(String klokkeslett, String beskrivelse) {
    this(null, klokkeslett, beskrivelse);
  }

  public boolean iDag() {
    // Kan nå sammenlikne this.dato med LocalDate.now()
    return this.dato == null || this.dato.equals(LocalDate.now());
  }

  public void skrivUt() {
    System.out.println(this.beskrivelse + " (" + this.klokkeslett + ")");
  }
}
```