Oversikt over IN1010
og
Java for deg som kan Python

Leif Harald Karlsen

Universitetet i Oslo

16.01.24

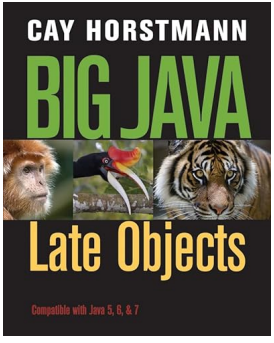


Plan

- Først: Praktisk informasjon og oversikt over kurset
- Resten av uken og neste mandag: Python for deg som kan Java
  - Skal kunne grunnleggenede Java-programmering
  - Altså, skrive enkle programmer tilsvarende det du kan i Python
  - Kunne Java tilsvarende kap. 1-8 i pensumboken
- Resten av semesteret: Objektorientert programmering mm.

Pensum

- Forelesningene og foilene til kurset
  - Publisert på timeplanen på semestersiden
  - Foiler/screencast fra forelesningene blir også publisert
- Ukesoppgavene og innleveringsoppgavene
  - Gjennomgås på gruppetimene og plenumsregningene
- Boken *Big Java, Late objects* av Hortsmann (2013)



Pensumliste på Leganto

`https://bibsys-k.alma.exlibrisgroup.com/leganto/readinglist/lists/19015363110002204?institute=47BIBSYS_UBO&auth=SAML`

Forelesere



Eric Bartley Jul



Eyvind Wærsted Axelsen



Leif Harald Karlsen

Andre nøkkelpersoner



Annika Willoch Olstad (lektor)



Ahmed Waseem Saeed (plenumsregner)

Gruppelærere og rettere

- Felicia Ediriweera Norén
- Haakon Persson Korslund
- Jonas Berger Nyvold
- Lilu Zhan
- Peter Alexander Døviken Løvold
- Robert Egelund
- Sigmund Håkonsen Trutt
- Synnøve Birkeland
- Agnes Hunstad Birkemo
- Andreas Askim Vatne
- Harald Flemmos Larsen
- Are Digranes
- Hanan
- Ingvild Gurine Kristiansen Utvik
- Jørund Holm Tangen
- Kenny Vu
- Leo Aleksander Sponnich
- Mikkel Alvegg-Langhelle
- Sander Segbø Løvaas
- Sofie Larssen Markeset
- William Henrik Behn
- Yuanyuan Jiang
- Ole Martin Henning
- Sander Halvorsen
- Siri Eikeland

Midtveiseksamen og eksamen

- Midtveiseksamen:
  - Tid: 18. mars 09:00 - 11:00 (2 timer)
  - Sted: Silurveien
  - Hjelpemidler: Ingen
  - Teller 25% av endelig karakter
  - Må bestås for å kunne gå opp til eksamen

- Avsluttende eksamen:
  - Tid: 5. juni 09:00 - 13:00 (4 timer)
  - Sted: Silurveien
  - Hjelpemidler: Alle trykte og skrevne
  - Teller 75% av endelig karakter

Obliger og innleveringsoppgaver

- 7 innleveringsoppgaver med tilbakemelding
- Levers i Devilry
- 2 av disse er obligatoriske(!)
- Første oblig må bestås for å kunne ta midtveiseksamen
- Siste oblig (og midtveis) må bestås for å kunne ta avsluttende eksamen

`https://www.uio.no/studier/emner/matnat/ifi/IN1010/v24/Obligatoriske-innleveringer`

Discourse

`https://astro-discourse.uio.no/c/in1010-24v`

(Ikke oppe ennå, men gir beskjed på semestersiden)

Undervisningen i IN1010

- Forelesning (tirsdag): Nytt stoff
- Gruppetimer: Utdype og øke forståelsen
- Plenumsregning (mandag): Repetisjon, eksempler
- Selvstendig arbeid: Ukesoppgaver, obliger, Trix, bøker, notater, foiler, forum...

- Husk: Forventes at dere jobber 13,5 timer hver uke med IN1010
- Programmering læres ved å programmere: Skriv masse kode!

Et par ord om læring

- Delta på undervisningen, gjør alle oppgaver, les tilbakemeldinger, osv.
- Husk: dere er er for å lære, ikke for å løse problemer!
- En av grunnene til endret oblig-opplegg
- Sjekk egen forståelse, og vær ærlige med dere selv
- Denne muligheten til å virkelig lære noe dypt får dere sjeldent igjen!

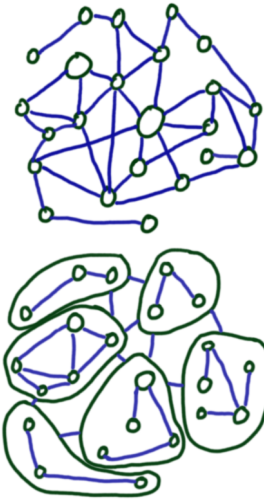
Dette skal du lære i IN1010

- Programmeringsspråket Java
- Avanserte objektorienterte mekanismer (subklasser, polymorfi, interface)
- Noen grunnleggende datastrukturer og deres implementasjon
- Løse reelle problemer av middels størrelse
- Utvikle robuste og pålitelige programmer med godt objektorientert design
- Lage parallelle programmer med tråder
- Hendelseshåndtering

`https://www.uio.no/studier/emner/matnat/ifi/IN1010/v24/timeplan/index.html`

Hvorfor lære objektorientert programmering?

- For å lage store og avanserte programmer som løser komplekse problemer må man kunne:
  - Dele programmet opp i flere mindre komponenter som løser hver sin del av problemet (hierarisk)
  - Komponentene må dele data/funksjonalitet viktig for andre komponenter
  - Komponentene må sjule detaljer som ikke er viktig for andre komponenter
  - Komponentene bør være gjenbrukbare på ulike variasjoner av problemet
- Objektorientering er én måte å gjøre dette på, hvor komponentene er objekter
- Objektorientering er en av de vanligste måtene å gjøre dette på
- Merk: Mye av pensum først nyttig når man skriver store programmer



Objektorientert programmering: Historie

- Første objektorienterte språket var Simula (1960-tallet)
- Ble laget av Ole Johan Dahl og Kristen Nygaard (på Norsk Regnesentral)
- Simulas objektorientering har inspirert mange av de mest populære språkene
  - C++, Java, C#, Object Pascal, Smalltalk, osv.





Ole Johan Dahl og Kristen Nygaard

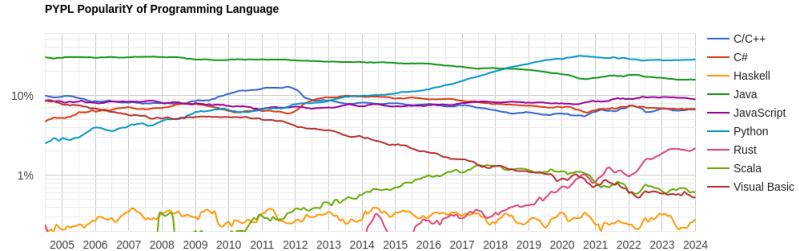
Forkunnskaper

Vi forventer at du kan programmere i Python vha.:

- Variabler, kostanter og uttrykk
- Tilordninger
- Kontrollstrukturer (løkker, if-else, osv.)
- Lister og ordbøker
- Metoder/funksjoner
- Klasser, objekter og referanser



Forfor lære flere programmeringsspråk?



- Ulike språk har ulike styrker, svakheter, fokusområder
- Gjerne laget for å tilfredstille et behov
- Det finnes altså ikke et beste programmeringsspråk!
- Som informatiker/utvikler bruker man ofte mange språk
- For hvert språk man lærer blir neste enklere å lære
- Merk: De fleste programmeringsspråk *kan* gjøre det meste
- Betyr ikke at det er best egnet for dette formålet

Ulike språk: Python



- Enkelt å lære og å lese
- Enkelt å skrive små programmer/scripts
- God støtte for numeriske beregninger (maskinlæring, statistikk, osv.)

Ulike språk: C



- Lavnivå: Full kontroll på ressursbruk
- Svært effektivt
- Krever mer av programmereren
- Egner seg godt for begrenset hardware eller applikasjoner som krever høy ytelse

Ulike språk: Java



- Kjører på en virtuell maskin, kryss-plattform
- Mellom-nivå: Lavere enn Python men høyere enn C
- Generelt høyere ytelse enn Python, lavere enn C
- Skreddersydd for objektorientering
- Egner seg godt for større programmer, Android-applikasjoner og desktop-programmer

Java vs. Python: Hovedforskjeller

Python:

- Dynamisk typet
  - Variable har ingen spesifisert type
  - Variable kan inneholde ulike typer under kjøring
  - Funksjoner/metoder kan ta og returnere verdier av ulike typer
- Tolket
  - Oversetter kode til maskin-instruksjoner under kjøring
  - Type-feil oppdages under kjøring

Java:

- Statisk typet
  - Variable har en spesifisert type
  - Variable kan kun inneholde verdier av den typen
  - Funksjoner/metoder har spesifisert argument- og retur-typer
- Kompilert
  - Kompilering: Oversetter all kode til maskin-instruksjoner først
  - Typefeil oppdages ved kompilering
  - Kompilering gir et optimert, type-sjekket program klar for kjøring

Fra Python til Java: Eksempel 1

Python (`hello.py`):

```
print("Hello, world!")
```

```
$ python3 hello.py
Hello, world!
```

Java (`Hello.java`):

```
class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
```

```
$ javac Hello.java
$ java Hello
Hello, world!
```

- All kode må inn i en klasse
- Filnavnet må være likt klassenavnet
- `main` kjøres når klassen kjøres
- `System.out.println` er printe-metoden
- `javac` brukes for å kompilere et Java-program (til en `.class`-fil)
- `java` brukes for å kjøre programmet (`.class`-filen)

Java vs. Python: Syntaks

Python:

```python
# Printer ut "Jepp" og "Hurra"
# hvis 1 er mindre enn 2
# Printer "Hadet!" til slutt


if 1 < 2:
    print("Jepp")
    print("Hurra")

print("Hadet!")
```

Java:

```java
// Printer ut "Jepp" og "Hurra"
// hvis 1 er mindre enn 2
// Printer "Hadet!" til slutt
class Test {
  public static void main(String[] args) {
    if (1 < 2) {
        System.out.println("Jepp");
        System.out.println("Hurra");
    }
    System.out.println("Hadet!");
  }
}
```

- Java bruker `//` for kommentar (fremfor Pythons #)
- Java bruker krøllparenteser (`{` og `}`) femfor innrykk for å markere kodeblokker
- Java bruker semikolon for skille kommandoer (fremfor linjeskift i Python)
- Java bryr seg ikke om innrykk og linjeskift
- Bruker likevel innrykk og linjeskift for lesbarhet

Java vs. Python: Typer og verdier

Python:

```
x = 1
y = x + 2.1
x = 2
s = "Hello!"
b = True
x = None
```

Java:

```
int x = 1;
double y = x + 2.1;
x = 2;
String s = "Hello!";
boolean b = true;
x = null;
```

- Alle variabler er typet i Java
- Typen spesifiseres før variabelnavnet når variabelen introduseres
- `null` brukes for ingen verdi (som Pythons `None`)

Basistyper i Java

Type	Verdier	Størrelse i bit	Initialiseres som
`byte`	Heltall mellom -2^7 og $2^7 - 1$	8	`0`
`short`	Heltall mellom -2^{15} og $2^{15} - 1$	16	`0`
`int`	Heltall mellom -2^{31} og $2^{31} - 1$	32	`0`
`long`	Heltall mellom -2^{63} og $2^{63} - 1$	64	`0L`
`float`	32-bits flyttall	32	`0.0f`
`double`	64-bits flyttall	64	`0.0d`
`boolean`	`true` eller `false`	8	`false`
`char`	Enkel karakter (f.eks. `'a'`)	16	`'\u000'`
`String`[1]	Tekstrenger av vilkårlig lengde	-	`null`

[1] Egentlig ikke en basistype, men oppfører seg som en.

Java vs. Python: If-tester

Python:

```python
x = 1
if x == 1:
    print("Den er 1.")
elif x == 2:
    print("Den er 2.")
elif x == 3:
    print("Den er 3.")
else:
    print("Vet ikke.")
```

Java:

```java
int x = 1;
if (x == 1) {
    System.out.println("Den er 1.");
} else if (x == 2) {
    System.out.println("Den er 2.");
} else if (x == 3) {
    System.out.println("Den er 3.");
} else {
    System.out.println("Vet ikke.");
}
```

- Må ha parentes rundt if-testen
- Bruker `else if` fremfor `elif`
- Husk: Krøllparenteser rundt kodeblokker

Fra Python til Java: Eksempel 2

Python:

```python
x = 5
m = None

if x > 5:
    m = "Oi, stort tall!"
elif x == 5:
    m = "Det var et passe tall!"
else:
    m = "Huff da, det var lavt!"

print(m)
```

Java:

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

    int x = 5; // Alle variable må ha type
    String m;  // Blir NULL

    if (x > 5) { // Husk parenteser i if-testen
      m = "Oi, stort tall!";
    } else if (x == 5) { // Husk, ingen elif
      m = "Det var et passe tall!";
    } else {
      m = "Huff da, det var lavt!";
    }
    System.out.println(m);
  }
}
```

Java vs. Python: While

Python:

```python
x = ""
i = 0
while i < 10:
    x = x + "."
    i += 1

print(x)
```

Java:

```java
String x = "";
int i = 0;
while (i < 10) {
  x = x + ".";
  i++;                  // eller i += 1;
}
System.out.println(x);
```

- Må ha parentes rundt testen
- `i++` øker `i` med `1`

Fra Python til Java: Eksempel 3

Python:

```python
n = 5

fak = 1
i = n   # Lag kopi

while i > 0:
    fak = fak * i
    i -= 1

print(str(n) + "! = " + str(fak))
```

Java:

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

    int n = 5;

    int fak = 1;
    int i = n;   // Lag kopi

    while (i > 0) {
      fak = fak * i;
      i--;       // eller i -= 1
    }
    System.out.println(n + "! = " + fak);
  }
}
```

Java vs. Python: For-løkke

Python:

```python
x = ""
for i in range(0, 10):
    x = x + "."

print(x)
```

Java:

```java
String x = "";
for (int i = 0; i < 10; i++) {
  x = x + ".";
}
System.out.println(x);
```

- En for-løkke har 3 deler separert med semikolon:
  1. Initialisering av telle-variabel (`int i = 0;`), kjøres før løkka starter
  2. Test som avgjør når løkka skal fortsette (`i < 10;`), avbryter når testen blir `false`
  3. Inkrementering som kjøres etter hver gjennomgang av løkka (`i++`)

Fra Python til Java: Eksempel 4

Python:

```python
n = 5
fak = 1

for i in range(n, 0, -1):
    fak = fak * i

print(str(n) + "! = " + str(fak))
```

Java:

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

    int n = 5;
    int fak = 1;

    for (int i = n; i > 0; i--) {
      fak = fak * i;
    }
    System.out.println(n + "! = " + fak);
  }
}
```

Oppsummering

	Python	**Java**
Koden	tolkes direkte	kompileres til `.class`-fil som kjøres
Kode som skal kjøres må legges	i en fil	i main-metode i en klasse
Kodeblokker markeres med	innrykk	`{` og `}`
Kommandoer separeres med	linjeskift	semikolon
Variabler/funksjoner/metoder er	utypet	typet
Kommentar starter med	`#`	`//`
Ingen verdi	`None`	`null`

If-test	`if x < 2:`
	`    ...`
	`elif x > 2:`
	`    ...`
	`else:`
	`    ...`
		`if (x < 2) {`
		`    ...`
		`} else if (x > 2) {`
		`    ...`
		`} else {`
		`    ...`
		`}`

While-løkke	`x = 0`
	`while x < 10:`
	`    ...`
	`    x += 1`
		`int x = 0;`
		`while (x < 10) {`
		`    ...`
		`    x++;`
		`}`

For-løkke	`for x in range(0, 10):`
	`    ...`
		`for (int x = 0; x < 10; x++) {`
		`    ...`
		`}`