Objektorienterad Programmering DAT043

Föreläsning 1
15/1 -18
Moa Johansson

Information

- Lärare: **Moa Johansson**
- Assistenter:
  - **Klara Granbom** **(lab mån)**
  - **Niklas Gustafsson** **(lab fre)**
  - **Elias Hällqvist** **(lab fre)**
  - **Jakob Wall** **(lab mån)**
- Se kurshemsida för mer information. Länk finns via Studieportalen (DAT043).

Presentera mig: Jag kommer hålla föreläsningar på mån+tisdag.

Presentera assistenterna: Dessa kommer hålla i labbar på mån em och fre em, samt övningar.

Information

- Studentrepresentanter:
  - **Ebba Håkansson**
  - **Mathias Lammers**
  - **Johannes Nilsson**
  - **Nora Ojensa**
  - **Nils Trubkin**
- Kontaktinformation finns på kurshemsidan.
- Är med och utvärderar kursen.
- Hör av er till någon av dem om ni vill lämna feedback på kursen (eller direkt till mig).

Upplägg

- 12 ordinarie föreläsningar planerade (mån + tis).
- 5 lärarledda övningstillfällen (ons).
  - Förbered er genom att försöka lösa uppgifterna i förväg.
  - Jobba gärna tillsammans - dela lösningar!
  - Lärare går igenom de svåraste enligt önskemål.
- 4 inlämningsuppgifter - obligatoriska.
  - Görs i grupper om två.
  - Diskutera gärna, men dela ej lösningar mellan grupper.
  - Kopierade lösningar = fusk —> avstängning…

12 föreläsningar. Detaljer kommer upp på hemsidan eftersom (beror lite på i vilken takt vi hinner med saker). I slutet blir det en samanfattning/tentaförberedelseföreläsning.

Om ni följer upplägget och gör övningsuppgifter och ser till att ni båda förstår era lösningar på inlämningarna så kommer tentaplugget bli lätt!
Spara inte allt till slutet av kursen.

Dela in er själva i grupper om två. Har ni ingen partner - matchmaking på dagens labb.
Skulle er labbpartner hoppa av eller försvinna - tala genast om för lärare!

Tenta

- **Skriftlig. Lördag 10/3.**
- Betyg på kurs bestäms av tentaresultatet.
- Två omtentor: 8/6 och 22/8.
- Glöm inte att registrera er.
- **Se Studentportalen för aktuell information.**

Något annorlunda format än tidigare år.
Del A: För att få godkänt / 3.
Del B: För dem som aspirerar på högre betyg 4-5.
Kommer bli tydligare vad som krävs för de olika betygsstegen, men i övrigt liknande frågor som tidigare år.

Dagens föreläsning

- Introduktion till Java.
- Objektorienterade programmeringsspråk.
- Imperativ programmering.
  - Satser, variabler, typer, uttryck i Java.

Intro till Java: det objektorienterad språk som denna kurs handlar om.

Demo: HelloWorld!

Java:

```
class HelloWorld {
 public static void main(String [] args) {
  System.out.println(”Hello World!”);
 }
}
```

```
$ javac HelloWorld.java
$ java HelloWorld
Hello World!
```

Haskell:

```
main = putStrLn ”Hello World!”
```

Notera: Java programmet är längre, mycket ”runtomkring” än Haskellprogrammet. Kommer att ta ett par föreläsningar innan vi har hunnit gå igenom all syntax ens i HelloWorld! Dock, notera likhet: main-metod/funktion är där programmet börjar exekvera.

Javas kompilator heter ”javac”. Kommer skapa en fil HelloWorld.class. Kommandot java HelloWorld kör denna.

Notera: semikolon avslutar rader i Java. Ej tabb-känsligt som Haskell. Notera även parenteser.

Javas infrastruktur

- Plattformsoberoende
- `javac` producerar **Java Bytecode**
  - e.g. `javac HelloWorld.java —> HelloWorld.class`
- Exekveras av ett run-time system: **Java Virtual Machine (JVM)**.
  - e.g. `java HelloWorld`
- Java Bytecode kan köras på alla datorer med **Java Runtime Enviroment (JRE)** installerat.
  - JRE = JVM + kompilerade Javabibliotek.

Java är plattformsoberoende. D.v.s. programmeraren behöver inte bekymra sig över vilket OS programmet ska köras på. Javas kompilator producerar ett slags högnivåmaskinkod som kallas Java Bytecode.

Om vi jämför med Haskell, så får man en stor körbar fil med runtime-systemet inbakat. För Java får man en betydligt mindre fil, med Java Bytecode, och för att köra programmet startar man JVM som tolkar det.

Eftersom Java introducerades -95 ungefär samtidigt som gemene man fick internet var detta en av ”säljargumenten”.

Objektorienterad programmering

Ur SAOL:

ob·jekt [–jek´t] substantiv *~et*; pl. ~
1 föremål; ibl. icke-person: *många intressanta objekt på auktionen*; *reduceras till ett objekt*
2 ⟨språkv.⟩ satsdel som **uttrycker föremål för handling**
– I sammansättn. *objekt-* (vanl. till *objekt 1*), *objekts-* (vanl. till *objekt 2*).

- ”Ada Lovelace programmerade den första datorn”.
- ”Programmet skriver ut ”Hello World” i terminalförstret”.
- ”Jag flyttar tornet två steg åt höger”.

Namnet ”objektorienterad programmering” kommer från språkvetenskapen, där det benämner en satsdel som är föremål för handling:
”Ada Lovelace programmerade den första datorn”. ”Programmet skriver ut ”Hello World” i terminalförstret”.

Alltså: Inom objektorienterad programmering struktureras programmen efter vilka objekt som vi vill modellera, och vad vi vill göra med dessa.

Introducerades runt 1960, men blev populärt på 90-talet. Först tack vare C++ från 1983, som är en utökning av det populära språket C. Java är idag ett av de mest populära språken, t.ex. för utveckling av Android-appar, och introducerades 1995. Innehåller idag mängder av bibliotek för allt möjligt som man som programmerare kan använda sig av.

Objektorienterad programmering

- **Objekt:** modeller av det programmet hanterar, e.g.
  - terminalfönster,
  - spelpjäser i ett schackprogram…
- **Klasser:** moduler i Java, innehållande, e.g.
  - Beskrivning av objekt
  - **Metoder:** vilka handlingar kan man utföra på objekten.
    - *jmfr. funktioner i Haskell.*

```
class HelloWorld {
 public static void main(String [] args) {
  System.out.println(”Hello World!”);
 }
}
```

Objekt är alltså ”modeller” av det programmet hanterar (e.g. meddelanden som skrivs ut, spelpjäser i ett schackspelsprogram).
Beskrivningar/definitioner av objekt tillsammans med *metoder* (handlingar man kan göra med/på) objekten (jmfr. med funktioner) samlas i Java i *klasser*.

I t.ex. Haskell är funktioner (d.v.s. det programmet gör) utgångspunkt. I Java och andra objektorienterade språk är det alltså objektet, det vi *gör något med.*

Notera att analogin inte stämmer till 100%. Klassen HelloWorld beskrev inget speciellt objekt. Däremot innehöll den en speciell metod (main) där Javaprogram alltid startar. Även println är en metod.

Vi återkommer till objektorientering lite senare i kursen. Inledningsvis ska vi tala om grunderna i Java och grundläggande syntax, och imperativ programmering.

Imperativ programmering

Ur SAOL:
1 im·per·at·iv [im`–] substantiv *~en ~er*
• uppmaningsform av verb t.ex. *köp* av *köpa*

I programmering:

• Sekvens av kommandon.
• Instruktioner i viss ordning. Kan ha sido-effekter.
  • *jmfr. do-notation i Haskell.*
• Variabler är normalt muterbara (kan ändras).
  • *jmfr. IORefs i Haskell.*

I imperativa programmeringsspråk skriver vi en sekvens av kommandon/instruktioner:
”Gör A, sen B, sen C osv…”

Sido-effekter (e.g. I/O) är tillåtna och kan ske var som helst i programmet (till skillnad från Haskell, där detta sker i do-notation).
Variabler är (normalt) muterbara så kan ändras.

Lägre grad av abstraktion från hur maskinkod och datorer fungerar. Nackdel: svårare att resonera om korrekthet.

Satser

- Imperativa program: sekvens av kommandon eller *satser.*
  - *Jmfr. uttryck av typ* `IO ()` *i Haskell.*
- Avslutas med semikolon.

```
System.out.println("x = " + x);
System.out.println("y = " + y);
```

```
do putStrLn $ "x = " ++ x
   putStrLn $ "y = " ++ y
```

Java: Imperativt: två kommandon efter varandra. Motsvarar Haskell's do-notation. Kan ha sidoeffekter. Satser innehåller uttryck.
Satsen själv har dock ingen typ, det har endast uttrycken som den består av.

(Lokala) Variabler

- Statiskt typat: Måste explicit ange variablers typer.
- Variabeldeklaration med eller utan initiering.
  - Förvalda värden om ej initierade, e.g. 0 för `int`.
  - OBS! Lokala variabler måste initieras innan användning, annars kompilerar inte koden.

```
int x = 5;
int y;
y = x+1;
boolean ärXStörre = x > y;
System.out.println(”x = ” + x);
System.out.println(”y = ” + y);
```

Primitiva typer

`byte`	8 bitars heltal
`short`	16 bitars heltal
`int`	32 bitars heltal
`long`	64 bitars heltal
`float`	32 bitars flyttal
`double`	64 bitars flyttal
`boolean`	true/false (förvalt false)
`char`	16 bitars Unicode-tecken

Dessa är de (vanligaste) sk. primitiva typerna i Java. Notera att de börjar med liten bokstav.
Flyttal = decimaltal.
Heltalen kan vara negativa (signed).
Förvalt värde (om du inte initierat din variabel) är 0 för de numeriska typerna och false för booleans. Har dock ingen betydelse för lokala variabler, som vi sett hittills. Men har betydelse senare när vi ska börja definiera objekt.
Beroende på hur stora numeriska värden ditt program ska hantera, väljs typ. Oftast använder vi dock int för heltal och double för decimaltal.

Uttryck

- Liknar Haskell. Precedens och associativitet avgör gruppering.
- Parentes kan annars användas.

Literaler	`5, -17, 'k'`
Variabler	`x, y, ärXStörre`
Aritmetiska operatorer	`+, -, *, /, %` (modulo), `-` (unärt minus/negation)
Jämförelser	`!=` (inte lika med), `==, <, >, <=, >=`
Logiska operatorer	`&&` (och), `||` (eller), `!` (icke)

Uttryck har typer, till skillnad från satser. Kom ihåg: satser består av uttryck.

Uttryck med sidoeffekter

- I imperativa språk kan all uttryck ha sidoeffekter, t.ex.
  - `x = 22;` ändrar värdet på variabeln `x` till `22`.
  - `x += 10;` är samma sak som `x = x + 10`. Även: `-=, *=.`
  - `++x;` öka värde på x med 1, returnera detta värde. Även `--x;`
  - `x++;` öka värde på x med 1, returnera ursprungligt värde. Även `x--;`

Vad skriver programmet ut?

```
int x = 10, y = 10;
System.out.println(x++ + ” ” + ++y);
System.out.println(x + ” ” + y);
```

Uttryck med sidoeffekter förekommer ofta som topp-nivå uttryck. Koden blir oftast enklare att läsa och förstå på det viset.
Men kan även förekomma i deluttryck (se andra raden in programmet). ++ och -- uttryck är vanliga när man har en variabel som fungerar som räknare (mer om detta när vi diskuterar loopar).

Demo: Exempel.java

Metodanrop

- Funktioner i Java kallas metoder.
- Statiska metoder (e.g. `main, Math.random`).
  - *`Klassnamn.metodnamn(parametrar)`*

```
double r = Math.random();
r = Math.ceil(r*10);
```

- Standardmetoder (e.g. `println`). Mer om dessa senare.
  - *`objekt.metodnamn(parametrar)`*

```
System.out.println(”Hello World”);
```

if-satser

if (e) s1 else s2

- Om det boolska uttrycket `e` är sant, exekveras `s1` annars `s2`.

```
if (slumptal <= 5){
     System.out.println("Talet är litet.");
   }
else
     System.out.println("Talet är stort.”);
System.out.println("Klart slut!");
```

- I Java är det tillåtet att utesluta else-delen.
  - Om `e` är falskt fortsätter programmet direkt med nästa sats istället.

```
if (slumptal <= 5){
     System.out.println("Talet är litet.");
   }
System.out.println("Klart slut!");
```

Demo: Exempel2.java

Vad utmärker Java?

	Rent funktionellt	Imperativt	Objektorienterat	Automatisk minneshantering
Haskell	x			x
Java		**x**	**x**	**x**
C#		x	x	x
C++		x	x	
C		x		

Automatisk minneshantering har nog de flesta högnivåspråk ni kommer använda. Det betyder att programmeraren inte behöver allokera och frigöra minne ”för hand”. Java har en s.k. ”garbage collector” som automatiskt frigör minne när något inte längre används av programmet. Priset är att programmet blir något lite långsammare än exempelvis samma program i C. Men det kan vara att föredra för att undvika memory-leaks och liknande svårhittade fel.

Java är även ett statiskt typat språk (till skillnad mot e.g. Python eller Javascript som är dynamiskt typade). Det betyder att variabler tilldelas sin typ vid kompilering, vilket kan vara bra för att upptäcka fel redan då.

Installera Java (JDK)

- Oracles Java-implementation: Java SE (Standard Edition).
- Finns för bl.a. PC, Mac, Linux.
- Installera JDK (Java Development Kit). Innehåller kompilator etc.
- Aktuell version är Java 9 (1_9_x).

Java API

- Javas API:
- https://docs.oracle.com/javase/9/docs/api/overview-summary.html
- Alla bibliotektsklasser, metoder och beskrivningar av dessa. MYCKET användbart.

Hemläxa! Titta på Javas API innan nästa lektion. Sök t.ex. gärna upp klassen Math, som vi såg i Exempel 2. Kolla vad det finns för andra matematiska metoder definierade där.

Utvecklingsmiljöer

IDE: (Integrated Development Environment): Mer än att editera programtext.

- Hantera många filer som ingår i samma projekt.
- Kompilera med knapptryck.
- Syntaxkontroll medan du skriver.
- Auto-completion.
- Automatiskt importera nödvändiga paket, hitta standardfunktioner.
- Versionshantering.
- … och mycket mer

Rekommenderat IDE för kursen är Eclipse.

Det finns även andra IDE's för Java (NetBeans, IntelliJ). De flesta labbhandledarna på kursen kommer fokusera på Eclipse.

Passa på att installera Java och Eclipse eller annat IDE på dagens labb, och testa dina första Javaprogram.