Dagens tema: Mer av det dere trenger til del 1

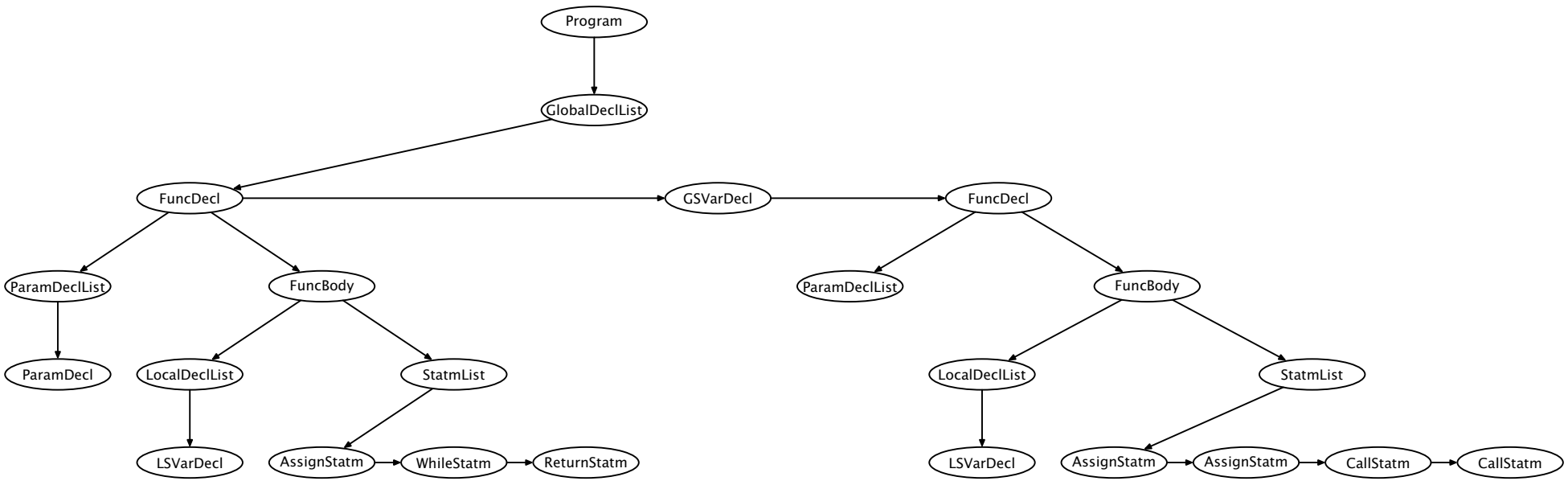
- Hvilke klasser trenger vi?
- Uttrykk
- Typer
- Versjonskontroll

Prosjektet

Utifra dette C♭-programmet:

```
int pot2 (int x)
{
   int p2;   p2 = 1;
   while (2*p2 <= x) { p2 = 2*p2; }
   return p2;
}

int x;

int main ()
{
   int v;   v = getint();
   x = pot2(v);   putint(x);   putchar(10);
}
```

. . . skal vi bygge opp dette treet (som her er klippet):



Koblingen mellom grammatikken og klasser

Normalt skal det være én klasse for hvert metasymbol (definisjon) i grammatikken:

while-statm


```
class WhileStatm extends Statement {
    Expression test;
    StatmList body;
```

Ekstra klasser

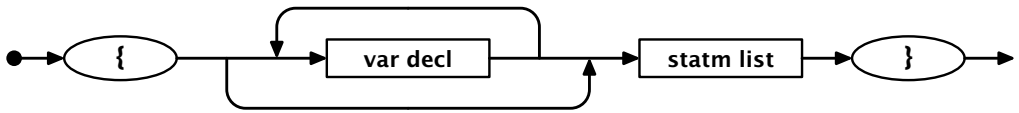
Noen ganger kan vi innføre ekstra klasser fordi

1. noen metasymboler hører naturlig sammen og/eller
2. vi må lagre noe informasjon.

Eksempel: Declaration

program

func body




```
abstract class Declaration extends SyntaxUnit {
    String name, assemblerName;
    Type type;
    boolean visible = false;
    Declaration nextDecl = null;
```

Noen ganger kan vi velge å implementere et metasymbol med flere klasser fordi

1. ulike varianter trenger forskjellig data og/eller
2. de kan trenge ulike virtuelle metoder.

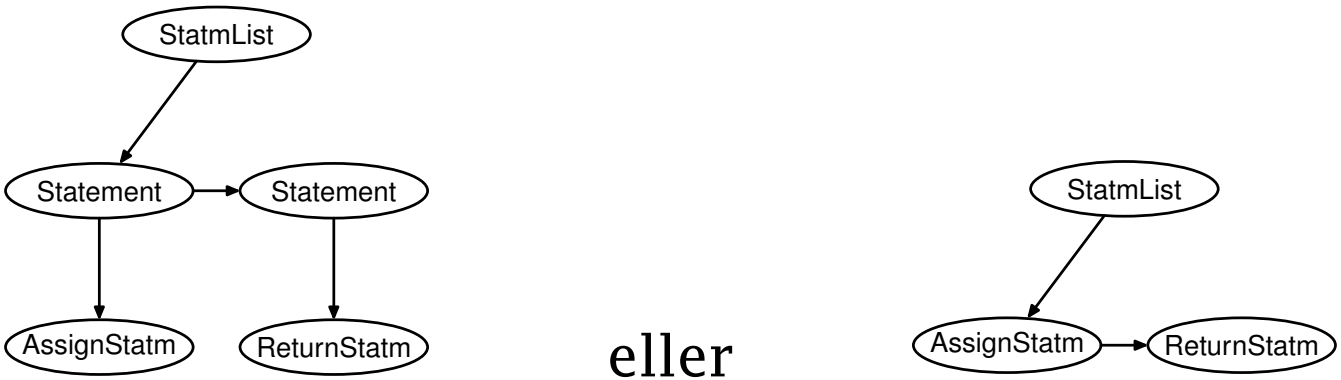
Eksempel: VarDecl

er implementert som en abstrakt klasse med fem underklasser: GlobalArrayDecl, GlobalSimpleVarDecl, LocalArrayDecl, LocalSimpleVarDecl og ParamDecl.

Noen ganger kan man droppe å lagre et metasymbol:

x = 4; return x;

kan lagres som



(Varianten til høyre er brukt i prosjektet vårt.)

Hint

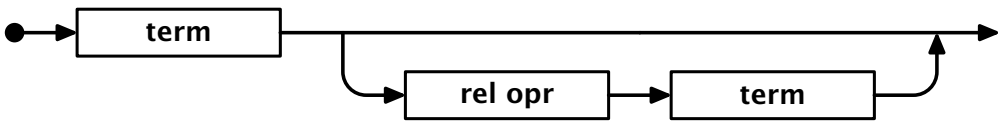
Hvis du stadig programmerer «Hvis det er den varianten, gjør jeg det, og hvis . . . », kan subklasse med virtuell metode være løsningen.

Dette gjelder spesielt ved test på instanceof.

Uttrykk

Et uttrykk har én eller to termer:

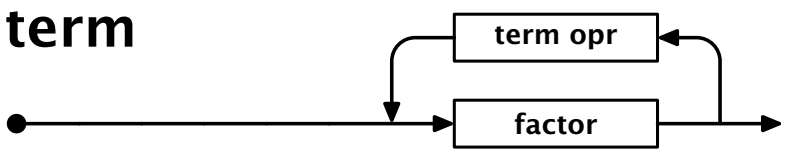
expression



```
class Expression extends Operand {
    Expression nextExpr = null;
    Term firstTerm, secondTerm = null;
    Operator relOp = null;
    boolean innerExpr = false;


    static Expression parse() {
        Log.enterParser("<expression>");

        Expression e = new Expression();
        e.firstTerm = Term.parse();
        if (Token.isRelOperator(Scanner.curToken)) {
            e.relOp = RelOperator.parse();
            e.secondTerm = Term.parse();
        }

        Log.leaveParser("</expression>");
        return e;
    }
```

Termer

En term har én eller flere faktorer:



Implementasjonen av dette må dere finne ut av selv.

Hvordan lagre uttrykk

```
[SyntaxUnit]
  ExprList
  [Operand]
    Expression
    FunctionCall
    Number
    Variable
  [Operator]
    RelOperator
  Term
```

- Disse klassene (og de du lager selv) brukes til å lagre uttrykk.
- Legg merke til at ⟨expression⟩ er en subklasse av ⟨operand⟩. Det er fordi vi kan ha indre uttrykk i parenteser (indikert av variabelen Expression.inner).

Typer

Modulen Types inneholder klasser og variabler for å representere typer. Basisklassen er Type:

Type.java

```
package no.uio.ifi.cflat.types;

public abstract class Type {
    public abstract int size();
    public abstract String typeName();

    public abstract void checkSameType(int lineNum, Type otherType, String what);
    public abstract void checkType(int lineNum, Type correctType, String what);
    public void genJumpIfZero(String jumpLabel) {}
}
```

Enkle type (dvs int og double) er instanser av BasicType.

BasicType.java

```
package no.uio.ifi.cflat.types;


public abstract class BasicType extends Type {
    @Override public void checkSameType(int lineNum, Type otherType, String what) {
        :
    }

    @Override public void checkType(int lineNum, Type correctType, String what) {
        :
    }

}
```

intType og doubleType opprettes som statiske variabler i modulen Types:

Types.java

```
package no.uio.ifi.cflat.types;

import no.uio.ifi.cflat.code.Code;
import no.uio.ifi.cflat.error.Error;
import no.uio.ifi.cflat.scanner.Token;
import static no.uio.ifi.cflat.scanner.Token.*;

public class Types {
    public static BasicType doubleType, intType;

    public static void init() {
        doubleType = new BasicType() {
                @Override public int size() {
                    return 8;
                }

                @Override public String typeName() {
                    return "double";
                }

                @Override public void genJumpIfZero(String jumpLabel) {
                }
            };
```

Anonyme subklasser

Vanligvis deklarerer vi klasser og subklasser slik:

Bil.java

```
abstract class Bil {
    abstract double antPassasjerer();
}
```

Sportsbil.java

```
class Sportsbil extends Bil {
    @Override double antPassasjerer() { return 1; }
}
```

Bilregister.java

```
class Bilregister {
    public static void main(String arg[]) {
        Bil a = new Sportsbil();
    }
}
```

Hvis vi ikke trenger Sportsbil til noe annet, kan vi oppnå tilsvarende med en anonym subklasse:

Bil.java

```
abstract class Bil {
    abstract double antPassasjerer();
}
```

Bilregister2.java

```
class Bilregister2 {
    public static void main(String arg[]) {
        Bil a = new Bil() {
                @Override double antPassasjerer() { return 1; }
            };
    }
}
```

```
$ ls *.class
Bil.class    Bilregister2$1.class    Bilregister2.class
```

Vektortyper

Vektorer får en type som er en ArrayType:

```
package no.uio.ifi.cflat.types;

public class ArrayType extends Type {
    public int nElems;
    public Type elemType;

    public ArrayType(int n, Type t) {
        nElems = n;  elemType = t;
    }

    @Override public int size() {
        return nElems*elemType.size();
    }

    @Override public String typeName() {
        return elemType.typeName() + " array";
    }

    @Override public void checkSameType(int lineNum, Type otherType, String what) {
    }

    @Override public void checkType(int lineNum, Type correctType, String what) {
    }
}
```

Når flere samarbeider

Når flere jobber sammen, kan man tråkke i beina på hverandre:

1. Per tar en kopi av en kildefil og begynner å rette på den.
2. Kari gjør det samme.
3. Kari blir første ferdig og kopierer filen tilbake.
4. Per blir ferdig og kopierer filen tilbake. Karis endringer går tapt.

Løsningen

Et *versjonskontrollsystem* er løsningen.

De fleste slike systemer er *utsjekkingssystemer* basert på *låsing*:

1. Per ber om og *sjekker ut* (dvs får en kopi av) filen og begynner å rette på den.
2. Kari ber om en kopi, men får den ikke fordi den er *låst*.

Først når Per er ferdig og *sjekker inn* filen, kan Kari få sin kopi.

Fordeler med et slikt utsjekkingssystem:

- Lettforståelig.
- Ganske sikkert.

  (Men hva om Per og Kari begge må rette i to filer hver? Da kan de starte med hver sin fil, men når de er ferdige med den første, finner de at den andre er sjekket ut.)

Ulemper:

- Kari bør kunne få en lese-kopi selv om Per jobber med filen. (Noen systemer tillater det, men ikke alle.)
- Hva om Per glemmer å legge tilbake filen?
- Det burde vært lov for Per og Kari å jobbe på ulike deler av filen samtidig.

Insjekkingssystemer

En bedre løsning er *innsjekkingssystemer*:

- Alle kan når som helst sjekke ut en kopi.
- Ved innsjekking kontrolleres filen mot andre innsjekkinger:
  - Hvis endringene som er gjort, ikke er i konflikt med andres endringer, *blandes* endringene med de tidligere.
  - Ved konflikt får brukeren beskjed om dette og må manuelt klare opp i sakene.

Et scenario

1. Per sjekker ut en kopi av en fil. Han begynner å gjøre endringer i slutten av filen.
2. Kari sjekker ut en kopi av den samme filen. Hun endrer bare i begynnelsen av filen.
3. Per sjekker inn sin kopi av filen.
4. Kari sjekker inn sin kopi, og systemet finner ut at de har jobbet på hver sin del. Innsjekkingen godtas.

Når man er alene

Selv om du jobber alene med et prosjekt, kan det være svært nyttig å bruke et versjonskontrollsystem:

- Man kan enkelt finne frem tidligere versjoner.
- Det kan hende man jobber på flere datamaskiner.

CVS og Subversion

Det mest kjente innsjekkingssystemet er **CVS** («Concurrent Versions System») laget i 1986 av *Dick Grune*. Det er spesielt mye brukt i Unix-miljøer.

For å bøte på noen svakheter i CVS laget firmaet *CollabNet* **Subversion** i 2000. Det ble en del av *Apache* i 2010.

Gratis implementasjoner finnes for alle vanlige operativsystemer; se http://subversion.apache.org/.

Nære og fjerne systemer

Subversion kan operere på to ulike måter:

- Alt skjer i det lokale filsystemet.
- Man kan starte en Subversion-tjener på en maskin og så sjekke inn og ut filer over nettet.

Vi skal gjøre det siste og bruke Ifis Subversion-tjener.

Opprette et *repository*

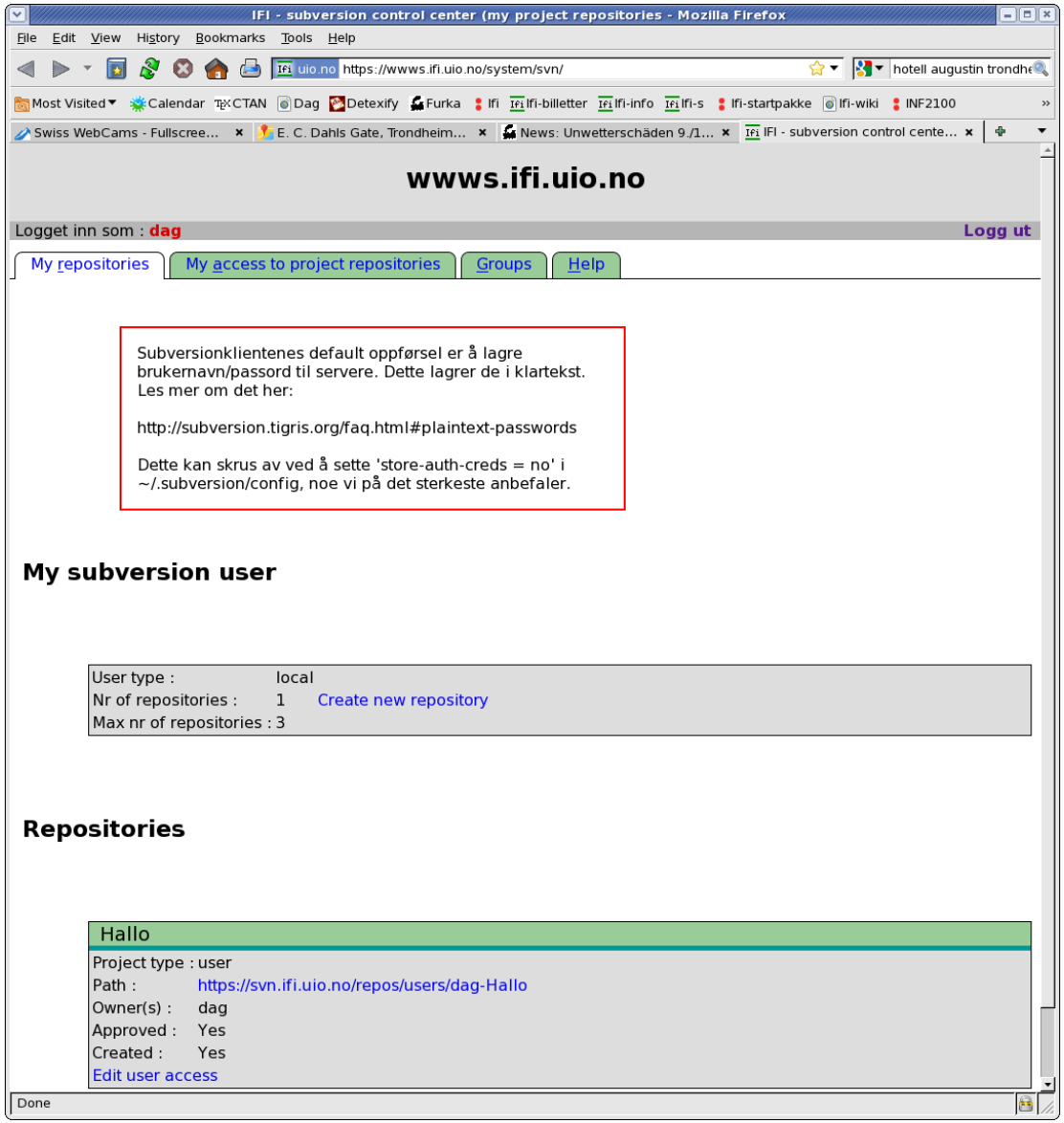
1. Gå inn på nettsiden

   https://wwws.ifi.uio.no/system/svn/

2. Logg inn.
3. Velg «My repositories» og «Create new repository». (I dette eksemplet heter det Hallo.)

   (Alle kan lage inntil tre «repositories».)
4. Hvis det er flere på prosjektet, velg «Edit user access».

Subversion


IFI - subversion control center (my project repositories - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://wwws.ifi.uio.no/system/svn/
Most Visited Calendar CTAN Dag Detexify Furka Ifi Ifi-billetter Ifi-info Ifi-s Ifi-startpakke Ifi-wiki INF2100
Swiss WebCams - Fullscree...
E. C. Dahls Gate, Trondheim...
News: Unwetterschäden 9./1...
IFI - subversion control cente...
wwws.ifi.uio.no
Logget inn som : dag
Logg ut
My repositories
My access to project repositories
Groups
Help
Subversionklientenes default oppførsel er å lagre brukernavn/passord til servere. Dette lagrer de i klartekst. Les mer om det her:
http://subversion.tigris.org/faq.html#plaintext-passwords
Dette kan skrus av ved å sette 'store-auth-creds = no' i ~/.subversion/config, noe vi på det sterkeste anbefaler.
My subversion user
User type : local
Nr of repositories : 1 Create new repository
Max nr of repositories : 3
Repositories
Hallo
Project type : user
Path : https://svn.ifi.uio.no/repos/users/dag-Hallo
Owner(s) : dag
Approved : Yes
Created : Yes
Edit user access
Done

Legge inn filer

Så kan vi legge inn mapper. La oss lage en *gren* med mappen Hei som inneholder filen Hello.java:

```
$ cd Hei
$ svn import https://sub.ifi.uio.no/repos/users/dag-Hallo -m "2100demo"
Adding         Hei/Hello.java

Committed revision 1.
```

Opsjonen -m gir en kort beskrivelse av denne grenen.

Sjekke ut filer

Nå kan vi (for eksempel fra en annen datamaskin) hente ut mappen vår:

```
$ svn co https://sub.ifi.uio.no/repos/users/dag-Hallo
A  dag-Hallo/Hello.java
Checked out revision 1.
$ ls -l
drwxr-xr-x    3 dag      ifi-a        4096 2011-11-13 06:46 dag-Hallo
$ ls -l -a dag-Hallo
total 16
drwxr-xr-x    3 dag      ifi-a        4096 2011-11-13 06:46 .
drwxr-xr-x    3 dag      ifi-a        4096 2011-11-13 06:46 ..
drwxr-xr-x    6 dag      ifi-a        4096 2011-11-13 06:46 .svn
-rw-r--r--    1 dag      ifi-a         500 2011-11-13 06:46 Hello.java
```

Sjekke inn filer

Etter at filen er endret, kan vi sjekke den inn igjen:

```
$ svn commit -m"Enklere kode"
Sending        Hello.java
Transmitting file data .
Committed revision 2.
```

Vi behøver ikke nevne hvilke filer som er endret — det finner Subversion ut selv. (Etter første utsjekking inneholder mappen skjulte opplysninger om repository-et, så det trenger vi ikke nevne mer.)

Andre nyttige kommandoer

svn update . henter inn eventuelle oppdateringer fra repository.

svn info viser informasjon om mappen vår:

```
$ svn info
Path: .
URL: https://sub.ifi.uio.no/repos/users/dag-Hallo
Repository Root: https://sub.ifi.uio.no/repos/users/dag-Hallo
Repository UUID: 8c927215-bc3e-0410-a56f-b2451114731f
Revision: 2
Node Kind: directory
Schedule: normal
Last Changed Author: dag
Last Changed Rev: 2
Last Changed Date: 2011-11-13 07:02:16 +0100 (Sun, 13 Nov 2011)
```

svn diff viser hvilke endringer som er gjort:

```
$ svn diff -r 1:2
Index: Hello.java
===================================================================
--- Hello.java  (revision 1)
+++ Hello.java  (revision 2)
@@ -7,10 +7,9 @@
     Properties prop = System.getProperties();
     String versjon = prop.getProperty("java.version"); // Versjonen
     String koding = prop.getProperty("file.encoding"); // Koding
-    String hei;
+    String hei = "Hallo";

-    hei = "Hallo";
-    hei = hei + ", alle sammen!";
+    hei += ", alle sammen!";
     System.out.println(hei);
     System.out.println("Dette er versjon " + versjon);
     System.out.println("Kodingen er " + koding);
```

Om man ikke vil sette opp en Subversion-tjener, finnes det mange (ofte gratis) alternativer på nettet, som Bitbucket, GitHub og andre.

Husk bare at dette prosjektet ikke får ligge åpent noe sted.

Konklusjon

De få timene man bruker på å lære seg et versjonskontrollsystem, betaler seg raskt.