

## Dagens tema:

- Kompilatorens struktur
  - Oppbyggingen
  - Pakker i Java
  - Avbrudd («exceptions»)
  - Enum-klasser i Java
- De ulike modulene
- Prosjektet
  - Hva skal **del-0** gjøre?
  - Feilmeldinger
  - Testutskrifter
  - Siste råd og påbud

## Prosjektet

Hvordan skriver man et større program som en kompilator?

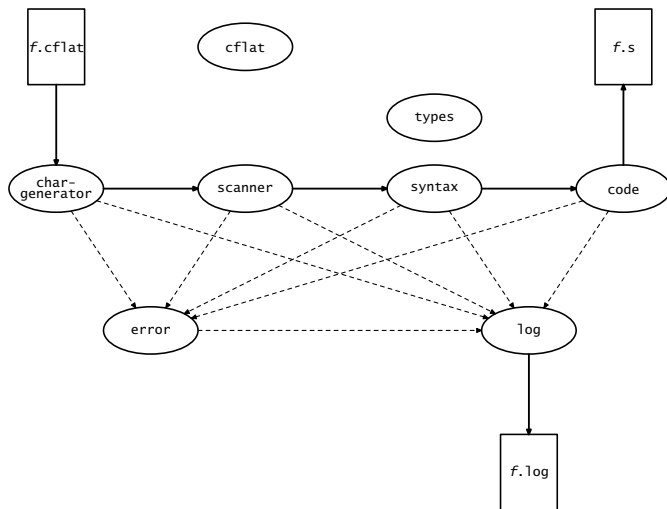
### Struktur

Det bør deles opp i passe store deler.

### Hvordan bør et program deles opp?

Ofte er det fornuftig å se på hvor data flyter.

## Oppdeling av programmer



Noen programmeringsspråk har mekanismer for store moduler – men langt fra alle. Java har package.

## Begrunnelse

Anta at vi skal utvikle et CAD-system. Et firma i India har laget en god GUI-modul.

**Men**, begge har en klasse Point.

Moduler kan løse dette problemet.

Alle filene som skal inngå i en Java-pakke starter med «package *navn*».

Pakkenavn bør bestå av opphavsstedets internettadresse (baklengs) og et lokalt navn. Vårt kompilatorprosjekt heter *no.uio.ifi.cflat*

## Eksempel

P1/A.java

```
package P1;  
  
public class A {  
    public static int x = 1;  
}
```

(Under kompileringen må klassen ligge i et fil-tre som tilsvarende leddene i pakkenavnet; våre filer ligger i

*no/uio/ifi/cflat/scanner/Scanner.java*

og tilsvarende.)

Vi kan hente klasser fra alle pakker så lenge de finnes i CLASSPATH:

B.java

```
class B {  
    public static void main (String arg[]) {  
        System.out.println("P1.A.x = " + P1.A.x);  
    }  
}
```

P1/A.java

```
package P1;  
  
public class A {  
    public static int x = 1;  
}
```

Ifis standard CLASSPATH er:

```
$ printenv CLASSPATH  
/local/opt/java/classes:  
/local/opt/java/classes/postgresql-8.3-603.jdbc4.jar:  
/hom/dag/java/classes:  
.
```

## Beskyttelse

Klasser kan beskyttes:

- er usynlig utenfor pakken.

**public** kan brukes fra andre pakker.

For klasselementer gjelder:

**private** er bare tilgjengelige i klassen.

**protected** er for klassen og subklasser.

- er bare for bruk innen pakken.

**public** kan benyttes overalt.



For å unngå å skrive mange lange navn som `no.uio.ifi.cflat.Cflat.version`, kan vi importere klasser fra pakker:

B.java

```
import P1.A;

class B {
    public static void main (String arg[]) {
        System.out.println("P1.A.x = " + A.x);
    }
}
```

P1/A.java

```
package P1;

public class A {
    public static int x = 1;
}
```

Vi kan også importere alle klassene fra en pakke:

B.java

```
import P1.*;

class B {
    public static void main (String arg[]) {
        System.out.println("P1.A.x = " + A.x);
    }
}
```

P1/A.java

```
package P1;

public class A {
    public static int x = 1;
}
```

En siste mulighet er å importere *statiske* deklarasjoner i en klasse:

B.java

```
import static P1.A.*;

class B {
    public static void main (String arg[]) {
        System.out.println("P1.A.x = " + x);
    }
}
```

P1/A.java

```
package P1;

public class A {
    public static int x = 1;
}
```

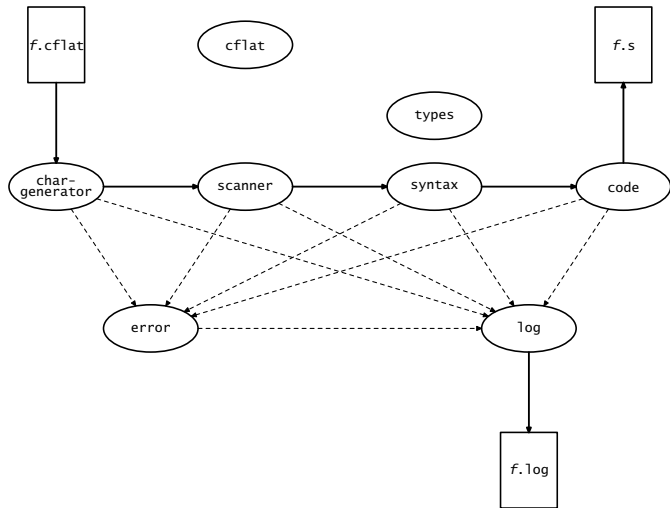
## Hva kan en pakke inneholde?

En Java-pakke kan bare inneholde klasser.

Vi trenger også data og metoder og vedtar derfor for prosjektet vårt:

- I alle pakker finnes en klasse med samme navn som pakken (men stor forbokstav); den inneholder data og metoder vi trenger, og alle er static.
- Alle disse klassene har disse to metodene:
  - init** benyttes til initiering av pakken.
  - finish** avslutter pakken.

Våre pakker



## Hovedprogrammet cflat

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

import java.io.*;
import no.uio.ifi.cflat.chargenerator.CharGenerator;
import no.uio.ifi.cflat.code.Code;
import no.uio.ifi.cflat.error.Error;
import no.uio.ifi.cflat.log.Log;
import no.uio.ifi.cflat.scanner.Scanner;
import static no.uio.ifi.cflat.scanner.Token.*;
import no.uio.ifi.cflat.syntax.Syntax;
import no.uio.ifi.cflat.types.Types;
```

## Modulen «cflat»

```
public class Cflat {
    public static final String version = "2013-08-30";

    public static String sourceName = null, // Source file name
        sourceBaseName = null;           // Source file name without extension
    public static boolean noLink = false; // Should we drop linking?
    public static final String myOS =     // The current operating system
        System.getProperty("os.name");

    private static boolean testParser = false, testScanner = false;
```

## Modulen «cflat»

```
private static boolean checkParams(String[] args) {
    for (String opt: args) {
        if (opt.equals("-c")) {
            noLink = true;
        } else if (opt.equals("-logB")) {
            Log.doLogBinding = true;
        } else if (opt.equals("-logP")) {
            Log.doLogParser = true;
        } else if (opt.equals("-logS")) {
            Log.doLogScanner = true;
        } else if (opt.equals("-logT")) {
            Log.doLogTree = true;
        } else if (opt.equals("-testparser")) {
            testParser = true;
            Log.doLogParser = Log.doLogTree = true;
        } else if (opt.equals("-testscanner")) {
            testScanner = true;
            Log.doLogScanner = true;
        } else if (opt.startsWith("-")) {
            return false;
        } else {
            if (sourceName != null) return false;
            sourceName = sourceBaseName = opt;
            if (opt.length()>3 && opt.endsWith(".cb"))
                sourceBaseName = opt.substring(0,opt.length()-3);
            else if (opt.length()>6 && opt.endsWith(".cflat"))
                sourceBaseName = opt.substring(0,opt.length()-6);
        }
    }
    return true;
}
```



## Modulen «cflat»

```
public static void main(String[] args) {
    int exitStatus = 0;

    if (checkParams(args) && sourceName!=null) {
        System.out.println("This is the Cb compiler (version " + version +
            " on " + myOS + ")");

        try {
            Error.init(); Log.init(); Code.init(); Types.init();
            CharGenerator.init(); Scanner.init(); Syntax.init();

            if (testScanner) {
                System.out.print("Scanning...");
                while (Scanner.nextNextToken != eofToken)
                    Scanner.readNext();
            } else {
                System.out.print("Parsing..."); Syntax.parseProgram();
                if (Log.doLogTree) {
                    System.out.print(" printing..."); Syntax.printProgram();
                }
                if (! testParser) {
                    System.out.print(" checking..."); Syntax.checkProgram();
                    System.out.print(" generating code..."); Syntax.genCode();
                }
            }
            System.out.println(" OK");
        }
    }
}
```



## Modulen «cflat»

```
        catch (RuntimeException e) {
            System.out.println();
            System.err.println(e.getMessage());
            exitStatus = 1;
        }
        finally {
            Syntax.finish(); Scanner.finish(); CharGenerator.finish();
            Types.finish(); Code.finish(); Log.finish(); Error.finish();
        }

        if (exitStatus==0 && ! testScanner && ! testParser)
            assembleCode();
    } else {
        System.err.println("Usage: cflat [-c] [-log{B|P|S|T}] " +
            "[-test{scanner|parser}] file");
        exitStatus = 2;
    }
    System.exit(exitStatus);
}
```

## Avbrudd

Noen ganger trenger vi å avbryte den normale utførelsen fordi en feil eller noe annet unormalt har skjedd. Hvis dette involverer mange metodekall, er avbrudd («exceptions») nyttig.

- **Sikre avbrudd** benyttes der vi vil være sikre på at feilen tas hånd om, f eks `FileNotFoundException`.
- **Kjøreavbrudd** krever ingen slik sikring.

## Avbrudd

```
void f(...) {  
    ⋮  
    throw new RuntimeException("Melding");  
    ⋮  
}
```

```
void g(...) {  
    try {  
        ⋮  
        f(...)  
        ⋮  
    } catch (RuntimeException e) {  
        // Hvis feil:  
        ⋮  
    }  
    finally {  
        // Alltid:  
        ⋮  
    }  
}
```

## Skanner

En kompilator *kan* lese og tolke en program tegn for tegn, men det er mye lettere om det kan *gjøres symbol for symbol*. Dette ordner en **skanner**.

**chargenerator** leser programkoden linje for linje, fjerner #-kommentarlinjer og deler de andre linjene opp i enkelt-tegn.

**scanner** fjerner /\*...\*/-kommentarer og setter tegnene sammen til symboler.

## Modulen «scanner»

```
/* Program som leser et tall v  
   og skriver ut v+1. */
```

```
# Hovedprogrammet:
```

```
int main ()  
{  
    int v;  
  
    /* Les data: */  
    v = getInt() + 1;  
    /* Skriv svaret: */  
    putInt(v);    putchar(10);  
}
```

har disse symbolene:

int	main	(	)	{	int	v	;	v	=	getInt	(	)	+
1	;	putInt	(	v	)	;	putchar	(	10	)	;	}	

## Enum-klasser

Noen ganger har man diskrete data som kun kan ha et begrenset antall fast definerte verdier:

**Kortfarge** Kløver, ruter, hjerter, spar

**Tippetegn** Hjemmeseier, uavgjort, borteseier

**Ukedag** Mandag, tirsdag, onsdag, torsdag, fredag,  
lørdag, søndag

Å representere disse med heltall er en halvgod løsning.

## Java tilbyr **enum**-klasser:

**Tippetegn.java**

```
enum Tippetegn {  
    Hjemmeseier, Uavgjort, Borteseier;  
}
```

Dette er **syntaktisk sukker** for noe à la

**Tippetegn.java**

```
class Tippetegn extends java.lang.Enum {  
    public static final Tippetegn  
        Hjemmeseier = new Tippetegn(),  
        Uavgjort = new Tippetegn(),  
        Borteseier = new Tippetegn();  
}
```



Slik brukes denne klassen:

**Tipping.java**

```
class Tipping {
    public static void main (String arg[]) {
        Tippetegn rekke[] = new Tippetegn[12+1];

        rekke[1] = Tippetegn.Hjemmeseier;
        rekke[2] = Tippetegn.Borteseier;
        rekke[3] = Tippetegn.Borteseier;

        for (int i = 1; i <= 3; ++i)
            System.out.print(rekke[i]+" ");
        System.out.println();
    }
}
```

```
> java Tipping
Hjemmeseier Borteseier Borteseier
```

## Hva kan vi gjøre med enum-klasser?

- Tilordne verdier («rekke[i] = Tippetegn.Uavgjort»)
- Sjekke på likhet og ulikhet («rekke[1] == Tippetegn.Borteseier»)
- Skrive ut objektet («System.out.println(rekke[1])» som er det samme som «System.out.println(rekke[1].toString())»)

## I vår skanner

Vår skanner kan levere følgende **token**:

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

```
public enum Token {  
    addToken, assignToken,  
    commaToken,  
    divideToken, doubleToken,  
    elseToken, eofToken, equalToken,  
    forToken,  
    greaterEqualToken, greaterToken,  
    ifToken, intToken,  
    leftBracketToken, leftCurlToken, leftParToken, lessEqualToken, lessToken,  
    multiplyToken,  
    nameToken, notEqualToken, numberToken,  
    rightBracketToken, rightCurlToken, rightParToken, returnToken,  
    semicolonToken, subtractToken,  
    whileToken;  
}
```

## Modulen chargenerator

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

import java.io.*;
import no.uio.ifi.cflat.cflat.Cflat;
import no.uio.ifi.cflat.error.Error;
import no.uio.ifi.cflat.log.Log;

public class CharGenerator {
    public static char curC, nextC;

    private static LineNumberReader sourceFile = null;
    private static String sourceLine;
    private static int sourcePos;
```

## Modulen «chargenerator»

```
public static void init() {
    try {
        sourceFile = new LineNumberReader(new FileReader(Cflat.sourceName));
    } catch (FileNotFoundException e) {
        Error.error("Cannot read " + Cflat.sourceName + "!");
    }
    sourceLine = ""; sourcePos = 0; curC = nextC = ' ';
    readNext(); readNext();
}

public static void finish() {
    if (sourceFile != null) {
        try {
            sourceFile.close();
        } catch (IOException e) {}
    }
}
```

Metoden `readNext` leser neste tegn; det blir lagt i `curC` og neste tegn i `nextC`.

```
public static void readNext() {
    curC = nextC;
    if (!isMoreToRead()) return;

    //-- Must be changed in part 0:
}
```

To nyttige metoder:

```
public static boolean isMoreToRead() {
    //-- Must be changed in part 0:
    return false;
}

public static int curLineNum() {
    return (sourceFile == null ? 0 : sourceFile.getLineNumber());
}
```

## Modulen scanner

Symbolene leses inn i curToken, nextToken og nextNextToken (samt i curName, curNumber, nextName, nextNumber, nextNextName og nextNextNumber).

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

import no.uio.ifi.cflat.chargenerator.CharGenerator;
import no.uio.ifi.cflat.error.Error;
import no.uio.ifi.cflat.log.Log;
import static no.uio.ifi.cflat.scanner.Token.*;

public class Scanner {
    public static Token curToken, nextToken, nextNextToken;
    public static String curName, nextName, nextNextName;
    public static int curNum, nextNum, nextNextNum;
    public static int curLine, nextLine, nextNextLine;

    public static void init() {
        //-- Must be changed in part 0:
    }

    public static void finish() {
        //-- Must be changed in part 0:
    }
}
```

## Lesingen skjer med readNext-metoden:

```
public static void readNext() {
    curToken = nextToken; nextToken = nextNextToken;
    curName = nextName; nextName = nextNextName;
    curNum = nextNum; nextNum = nextNextNum;
    curLine = nextLine; nextLine = nextNextLine;

    nextNextToken = null;
    while (nextNextToken == null) {
        nextNextLine = CharGenerator.curLineNum();

        if (! CharGenerator.isMoreToRead()) {
            nextNextToken = eofToken;
        } else
        //-- Must be changed in part 0:
        {
            Error.error(nextNextLine,
                "Illegal symbol: '" + CharGenerator.curC + "'!");
        }
    }
    Log.noteToken();
}
```



# Hva er en god feilmelding?

## Ubrukelig

```
ERROR: Syntax error detected!
```

## Noe bedre

```
ERROR: Syntax error found in line 217.
```

## Enda litt bedre

```
ERROR: Syntax error found in line 217:  
  x = x+1 } ;
```

## Melding med mening

Meldingen bør fortelle hva som er galt:

```
ERROR in line 217: Semicolon expected.  
  x = x+1 } ;
```

## Den beste meldingen

Meldingen bør angi hvorledes kompilatoren «tenker»:

```
ERROR in line 217:  
Expected ';' at end of sentence but found '}'.  
  x = x+1 } ;
```

## Feil

Hva gjør man med feil?

- Før prøvde man å finne så mange feil som mulig.
- Vi skal stoppe med melding ved første feil.

## Modulen error

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

import no.uio.ifi.cflat.log.Log;
import no.uio.ifi.cflat.scanner.Scanner;

public class Error {
    public static void error(String where, String message) {
        String eMess = "Cb error" +
            (where.length()>0 ? " "+where : "") + ": " + message;

        Log.noteError(eMess);
        throw new RuntimeException(eMess);
    }

    public static void error(String message) {
        error("", message);
    }

    public static void error(int lineNum, String message) {
        error((lineNum>0 ? "in line "+lineNum : ""), message);
    }
}
```

## Noen ganger tabber vi oss ut!

```
public static void panic(String where) {  
    error("in method "+where, "PANIC! PROGRAMMING ERROR!");  
}
```

## En nyttig rutine

```
public static void expected(String exp) {  
    error(Scanner.curLine,  
        exp + " expected, but found a " + Scanner.curToken + "!");  
}
```

Siden de fleste feilene er relatert til lesingen av  $C_b$ -koden, er det nyttig med en egen metode i scanner-modulen:

```
public static void check(Token t) {
    if (curToken != t)
        Error.expected("A " + t);
}

public static void check(Token t1, Token t2) {
    if (curToken != t1 && curToken != t2)
        Error.expected("A " + t1 + " or a " + t2);
}

public static void skip(Token t) {
    check(t);  readNext();
}

public static void skip(Token t1, Token t2) {
    check(t1,t2);  readNext();
}
```

## Modulen log

Brukeren kan slå av og på logging (med opsjoner som håndteres av cflat-modulen).

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

import java.io.*;
import no.uio.ifi.cflat.cflat.Cflat;
import no.uio.ifi.cflat.error.Error;
import no.uio.ifi.cflat.scanner.Scanner;
import static no.uio.ifi.cflat.scanner.Token.*;

public class Log {
    public static boolean doLogBinding = false, doLogParser = false,
        doLogScanner = false, doLogTree = false;

    private static String logName, curTreeLine = "";
    private static int nLogLines = 0, parseLevel = 0, treeLevel = 0;

    public static void init() {
        logName = Cflat.sourceBaseName + ".log";
    }

    public static void finish() {
        //-- Must be changed in part 0:
    }
}
```

## Testutskriften

Alle vil gjøre feil under arbeidet med kompilatoren. For enklere å oppdage feilene når de skjer, skal vi bygge inn ulike testutskriften som brukeren enkelt kan slå på:

Opsjon	Hva dumpes?	Del
-logB	Navnebindingen	2
-logP	Parseringen	1
-logS	Skanneren	0
-logT	Lagret parseringstre	1



## Modulen «log»

```
$ cflat -logS mini.cflat
```

```
$ more mini.log
```

```
1: # Program 'mini'
2: # -----
3: # A minimal Cb program!
4:
5: int main ()
Scanner: intToken
Scanner: nameToken main
Scanner: leftParToken
Scanner: rightParToken
6: {
Scanner: leftCurlyToken
7: putchar('x');
Scanner: nameToken putchar
Scanner: leftParToken
Scanner: numberToken 120
Scanner: rightParToken
Scanner: semicolonToken
8: }
Scanner: rightCurlyToken
Scanner: eofToken
```

- Siden utskriften på forrige side kommer fra to kilder, vil flettingen av den kunne variere.

## NB!

Variasjoner i fletting er helt normalt og må forventes.

- Når hele programmet er lest, vil skanneren bare returnere **eofToken**.

For å sikre loggfilen, må den lukkes etter hver utskrift.

```
private static void writeLogLine(String data) {
    try {
        PrintWriter log = (nLogLines==0 ? new PrintWriter(logName) :
            new PrintWriter(new FileOutputStream(logName,true)));
        log.println(data); ++nLogLines;
        log.close();
    } catch (FileNotFoundException e) {
        Error.error("Cannot open log file " + logName + "!");
    }
}
```

## I del 0 skal vi sjekke chargenerator og scanner:

```
public static void noteSourceLine(int lineNum, String line) {
    if (! doLogParser && ! doLogScanner) return;

    //-- Must be changed in part 0:
}

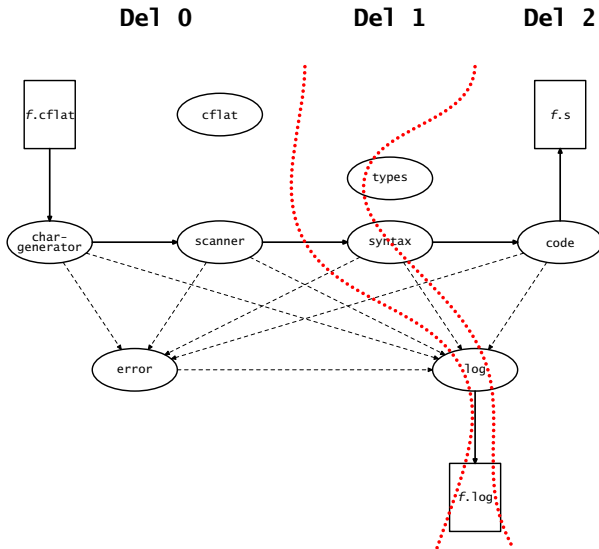
public static void noteToken() {
    if (! doLogScanner) return;

    //-- Must be changed in part 0:
}
```

## Feilmeldinger må med i loggen (om det er noen logg):

```
public static void noteError(String message) {
    if (nLogLines > 0)
        writeLogLine(message);
}
```

Del-0



## Siste innspill

- Del-0 skal fungere med opsjonen -testscanner.
- Les kompendiet!
- Les Java-koden! (Den største arbeidsinnsatsen ligger i å forstå denne koden.)
- Det er ikke lov å fjerne noe i basiskoden (men det er lov å legge til).
- Alt skal programmeres fra bunnen av. Det er altså ikke lov å bruke annet fra Java-biblioteket enn java.io (og \*Tokenizer er heller ikke lov).
- Gruppelærerene er der for å hjelpe dere.
- Begynn i tide!