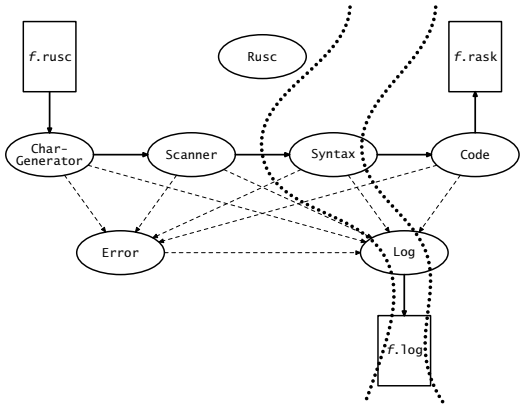
Dagens tema:

- Moduler i prosjektet
  - error
  - log
  - chargenerator
  - scanner
- Enum-klasser i Java
- Programmeringsverktøy
  - Emacs
  - Eclipse

Prosjektet

Del-0 **Del-1 Del-2**



Modulen error

Tidligere var det veldig viktig å finne så mange feil som mulig i et program. I dag er brukerne bare interessert i den første.

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


import no.uio.ifi.rusc.chargenerator.CharGenerator;
import no.uio.ifi.rusc.code.Code;
import no.uio.ifi.rusc.log.Log;
import no.uio.ifi.rusc.scanner.Scanner;
import no.uio.ifi.rusc.syntax.Syntax;

/**
 * Print error messages.
 */
public class Error {

    public static void init() {
        //-- Må endres i del 0:
    }
    public static void finish() {
        //-- Må endres i del 0:
    }
```

Hva er en god feilmelding? Den må i hvert fall identifisere linjen der feilen er.

```
public static void error(String where, String message) {
    //-- Må endres i del 0:

    Syntax.finish();  Scanner.finish();  CharGenerator.finish();
    Log.finish();   Code.finish();   Error.finish();
    System.exit(1);
}

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

public static void error(int lineNum, String message) {
    error("in line " + lineNum, message);
}

public static void giveUsage() {
    System.err.println("Usage: rusc [-log{C|P|S|T}] [-test{scanner|parser}] file");
    System.exit(2);
}
```

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

```
public static void illegal(String message) {
    Error.error(CharGenerator.curLineNum(), message);
}
```

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 + "!");
    }
}
```

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

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


import java.io.*;
import no.uio.ifi.rusc.error.Error;
import no.uio.ifi.rusc.rusc.Rusc;
import no.uio.ifi.rusc.scanner.Scanner;
import no.uio.ifi.rusc.scanner.Token;

/**
 * Produce logging information.
 */
public class Log {
    public static boolean doLogCode = 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 = Rusc.sourceName;
        if (logName.endsWith(".rusc"))
            logName = logName.substring(0, logName.length()-5);
        logName += ".log";
    }
```

I del 0 skal vi sjekke skanneren:

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

    //-- Må endres i del 0:
}

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

    //-- Må endres i del 0:
}
```

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

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

Modulen chargenerator

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


import java.io.*;
import no.uio.ifi.rusc.error.Error;
import no.uio.ifi.rusc.log.Log;
import no.uio.ifi.rusc.rusc.Rusc;

/**
 * Module for reading single characters.
 */
public class CharGenerator {
    public static char curC, nextC;

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


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

public static void finish() {
    if (sourceFile != null) {
        try {
            sourceFile.close();
        } catch (IOException e) {
            Error.error("Could not clse source file!");
        }
    }
}
```

Metoden readNext leser neste tegn:

```
public static boolean isMoreToRead() {
    //-- Må endres i del 0:
    return false;
}

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

    //-- Må endres i del 0:
}
```

Modulen scanner

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

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


import no.uio.ifi.rusc.chargenerator.CharGenerator;
import no.uio.ifi.rusc.error.Error;
import no.uio.ifi.rusc.log.Log;

/**
 * Module for forming characters into tokens.
 *
 */
public class Scanner {
    public static Token curToken, nextToken;
    public static String curName, nextName;
    public static long curNum, nextNum;
```

Lesingen skjer med readNext-metoden:

```
public static void readNext() {
    curToken = nextToken;  curName = nextName;  curNum = nextNum;
    nextToken = Token.noToken;
    while (nextToken == Token.noToken) {
        //-- Må endres i del 0:
    }
    Log.noteToken(nextToken);
}
```

Noen nyttige hjelperutiner:

```
public static void check(Token t) {
    if (curToken != t)
        illegal("Expected a " + t + " but found a " + curToken + "!");
}

public static void check(Token t1, Token t2) {
    if (curToken != t1 && curToken != t2)
        illegal("Expected a " + t1 + " or a " + t2 + " but found a " + curToken + "!");
}

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

Token-verdiene er av en **enum-klasse**.

Et annet eksempel er:

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

    char Tegn () {
        return toString().charAt(0);
    }
}
```

Dette er *nesten* det samme som

```
class Ttegn {
    public static final Ttegn Hjemmeseier = new Ttegn(),
        Uavgjort = new Ttegn(), Borteseier = new Ttegn();

    public String toString () {
        if (this == Hjemmeseier) return "Hjemmeseier";
        else if (this == Uavgjort) return "Uavgjort";
        else return "Borteseier";
    }

    char tegn () {
        return toString().charAt(0);
    }
}
```

Men vi får ikke lov å bruke new på dem.

Slik brukes denne klassen:

```
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].Tegn());
        System.out.println();
    }
}
```


Hva kan vi gjøre med enum-klasser?

- Tilordne verdier («rekke[i] = Tippetegn.Uavgjort»)
- Sjekke på likhet og ulikhet («rekke[1] == Tippetegn.Boreseier»)
- 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.rusc.scanner;

/*
 * class Token
 */

/**
 * The different kinds of tokens read by Scanner.
 */
public enum Token { addToken, assignToken, commaToken, divideToken, elseToken,
    eofToken, equalToken, funcToken, greaterEqualToken, greaterToken, ifToken,
    intToken, leftBracketToken, leftCurlToken, leftParToken, lessEqualToken,
    lessToken, multiplyToken, nameToken, notEqualToken, noToken, numberToken,
    rightBracketToken, rightCurlToken, rightParToken, returnToken,
    semicolonToken, subtractToken, whileToken;

    public static boolean isOperator(Token t) {
        //-- Må endres i del 0:
        return false;
    }
}
```


Slik henter man malen til prosjektet:

```
copy ~inf2100/oblig/base.zip .
unzip base.zip
cd inf2100/
make
```

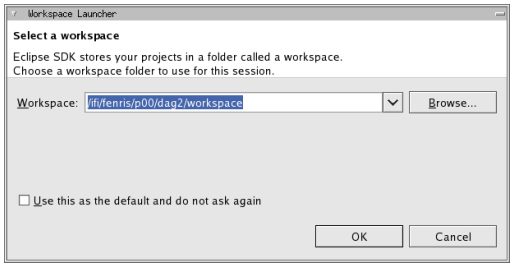
Hvilket programmeringsverktøy?

Det finnes to typer verktøy:

- **Generelle verktøy** (som Emacs) kan håndtere alle programmeringsspråk.
- **Spesialverktøy** (som Eclipse) er laget for ett programmeringsspråk (og ofte bare for én omgivelse).

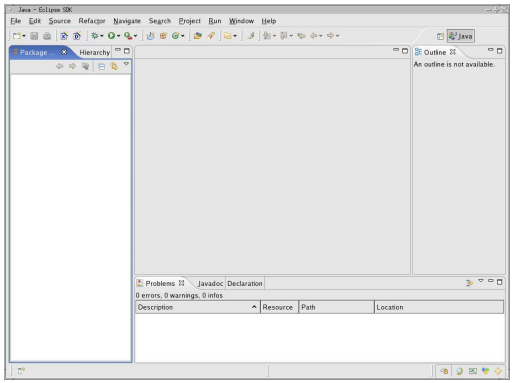
Oppstart av Eclipse:

> eclipse&

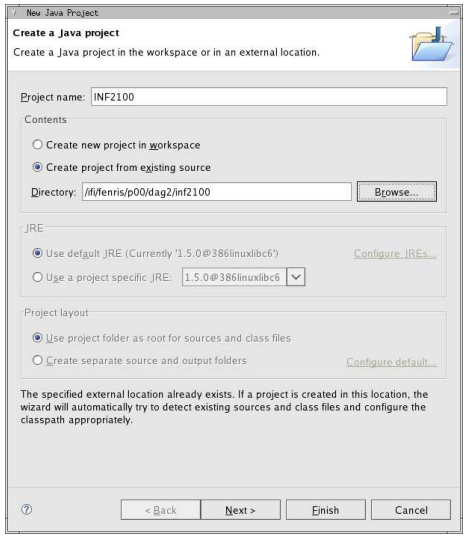


Velkomstvinduet i Eclipse

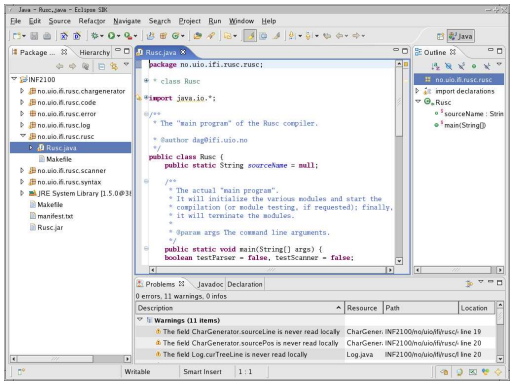
Eclipse-vinduet



Start et nytt prosjekt



Arbeid med Eclipse



Det er mange fordeler med et spesialverktøy:

- Mange liker å jobbe med «pek-og-klikk».
- Verktøyet vil oppdage feil veldig tidlig.
- Mye kode settes inn automatisk.
- Ved feil under testing får man øyeblikkelig vite hvor feil oppsto.

Hvorfor bruker da ikke alle spesialverktøy?

- Det er ikke laget spesialverktøy for alle språk og omgivelser.
- Spesialverktøy er grafiske – det er ikke alltid man kan kjøre slike programmer.
- Det er mye jobb å lære et nytt grafisk verktøy for hvert språk man skal jobbe i.
- Man jobber ofte raskere med tastaturet enn med musen.
- Noen nyttige operasjoner finnes ikke i spesialverktøyene (bytt to tegn, bytt om variabelnavn, . . .).

Konklusjon

Alle bør prøve begge typen verktøy.

Så velger man det man foretrekker til hvert prosjekt.