コンパイラ演習資料（javacc）　　担当：佐々木晃

□演習問題 B0（締め切り＝2009/06/15 演習中の回収時）

問題番号: B0
課題名：コンパイラの作成準備
出題日：06/08
課題：JavaCC の設定と準備
本演習資料にしたがって、中置記法による算術式（四則演算、括弧が使える）を逆ポーランドによる式に変換せよ。

（参考、コンパイラ作成１）
http://cis.k.hosei.ac.jp/~asasaki/lect/compiler/2008A/problem/problem1.htm

0. javacc について。

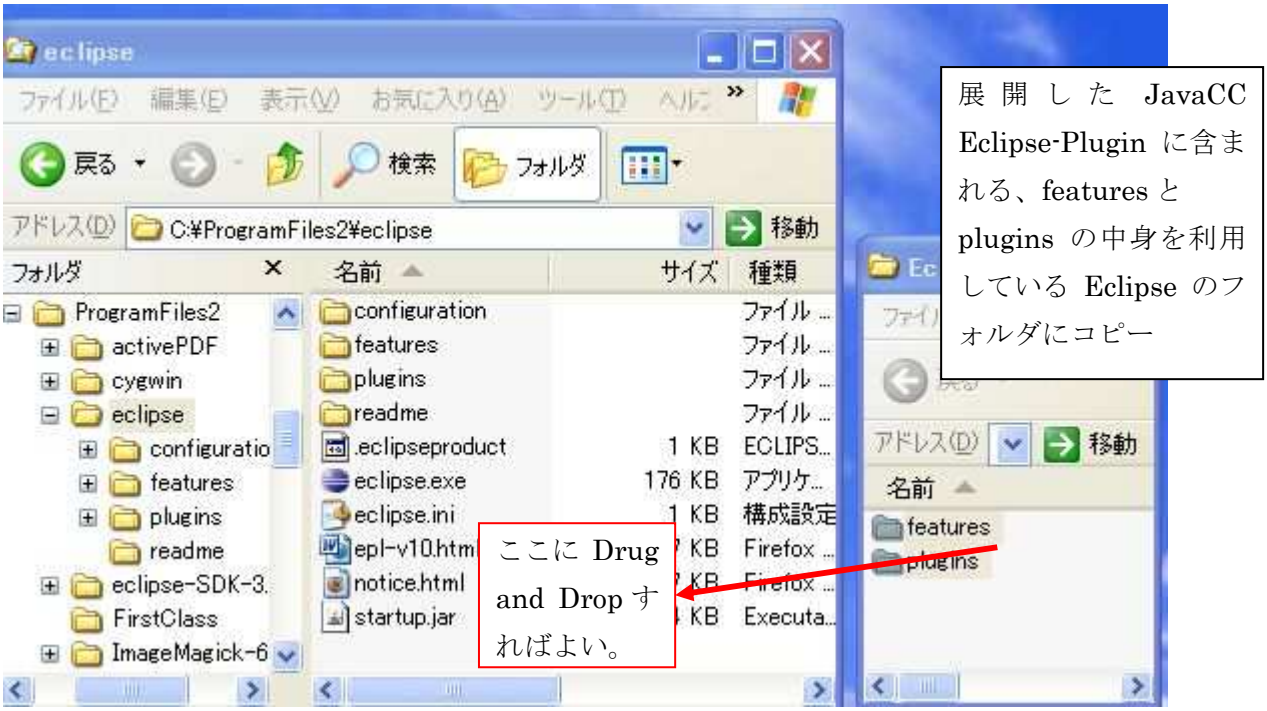
詳しくは、中田先生のページを参照
http://cis.k.hosei.ac.jp/~nakata/lectureCompiler/JavaCC/index.html
その他、http://cis.k.hosei.ac.jp/~asasaki/lect/compiler/の JavaCC の項目を参考にせよ。

1. javacc のインストール (Eclipse-plugin を利用)

説明は下記にある。
Eclipse JavaCC Plugin Home
http://eclipse-javacc.sourceforge.net/

下記からダウンロード
SourceForge.net: JavaCC Eclipse Plugin
http://sourceforge.net/projects/eclipse-javacc

インストール。アーカイブを展開（解凍）した中には、features と plugins というフォルダが含まれる。この中身を利用している Eclipse のルートフォルダにある features, plugins のそれぞれに入れてやればよい。

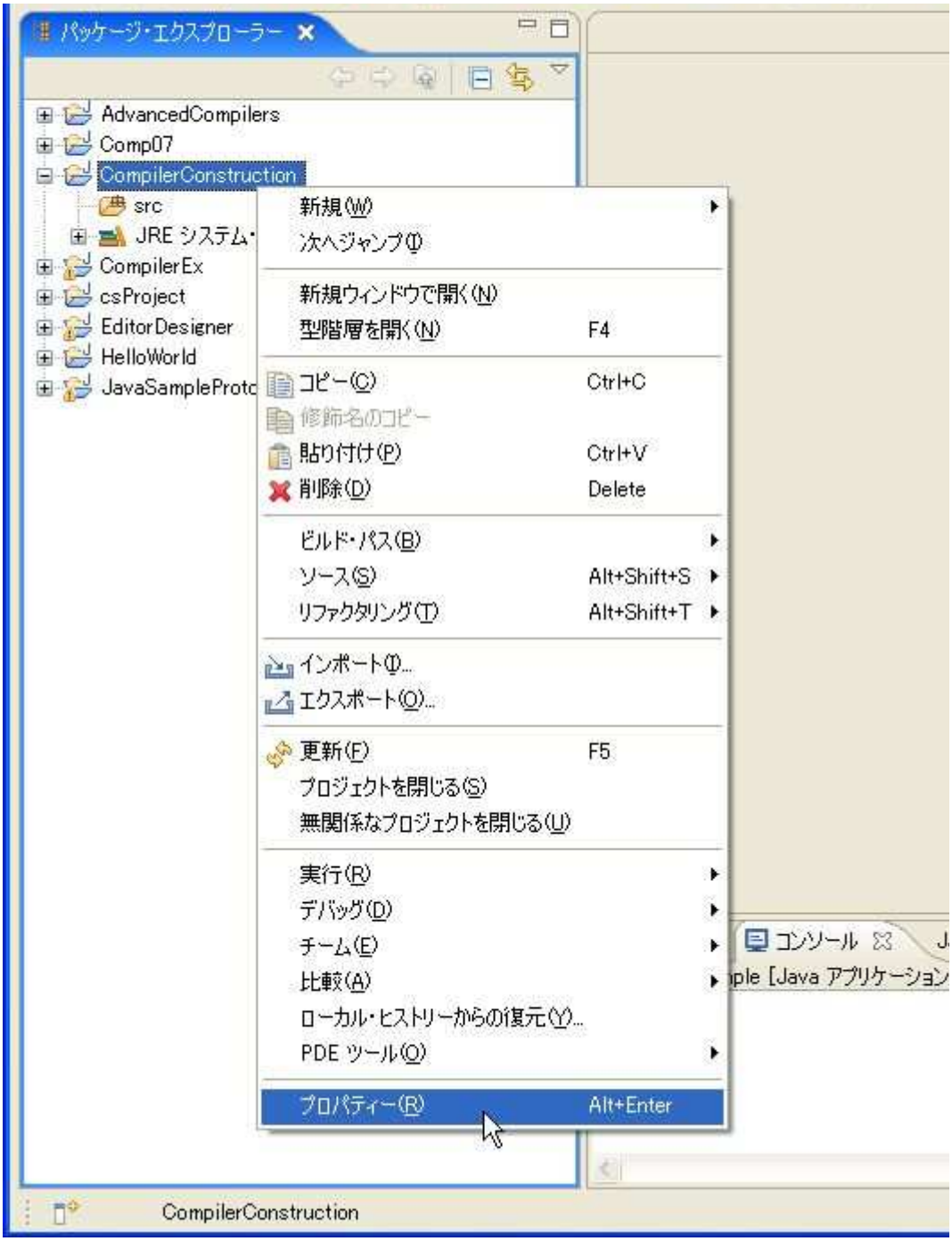


以上で終了

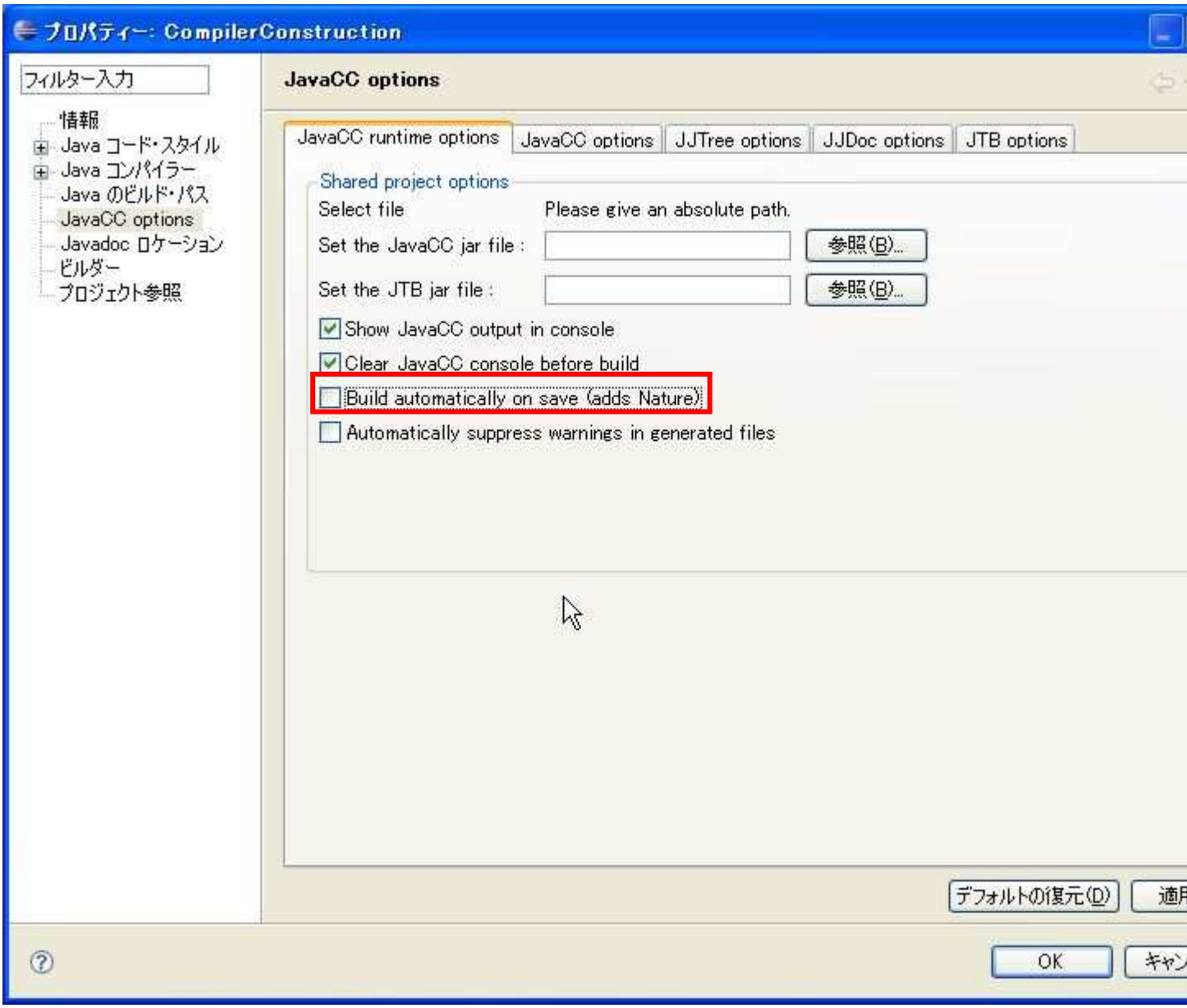
2. プロジェクトと JavaCC オプションの設定

ここでは CompilerConstruction というプロジェクトを設定。通常と同じく Java プロジェクトを指定。

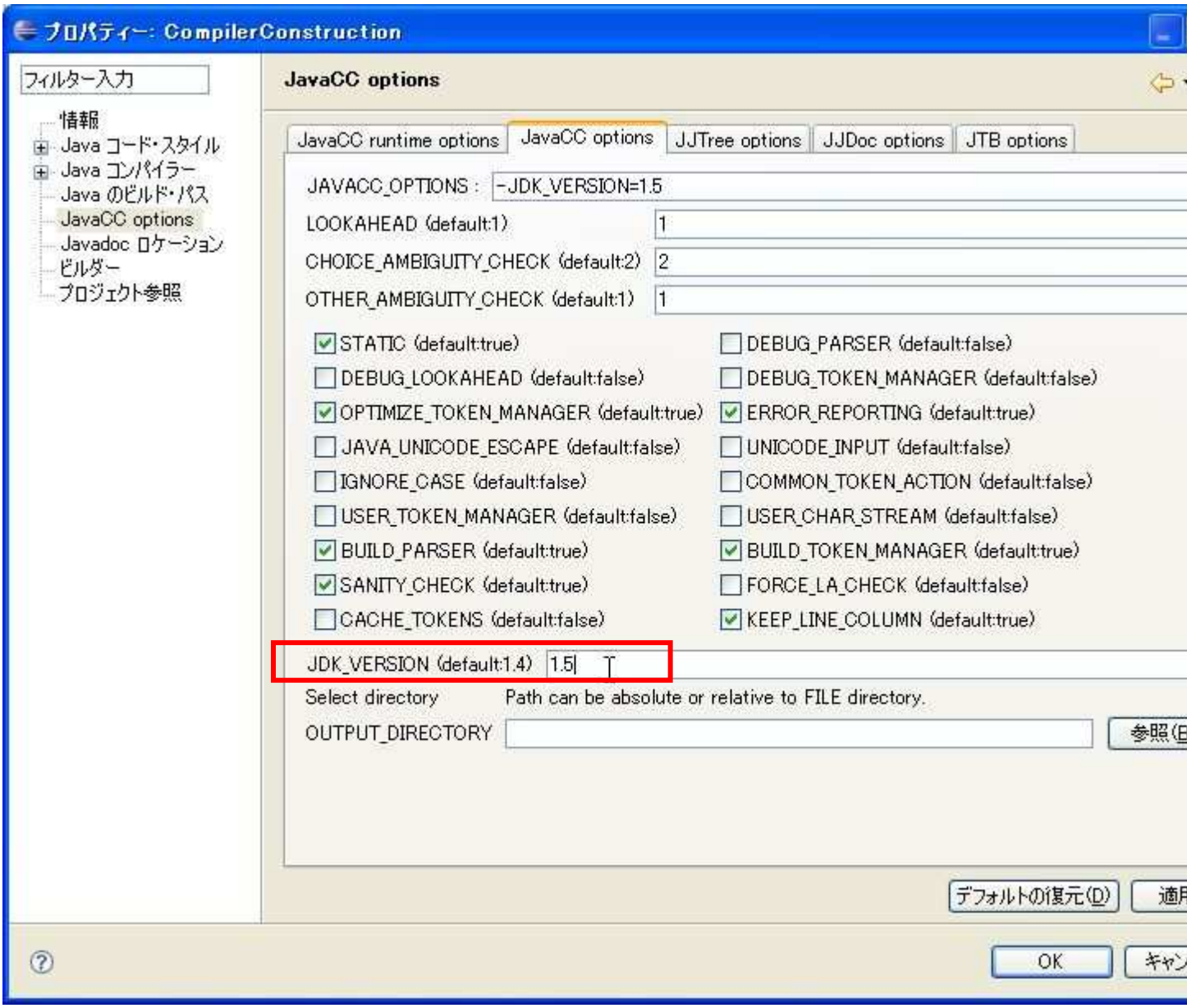
JavaCC のオプションの設定。



自動セーブ機能をＯＦＦ(好みで)



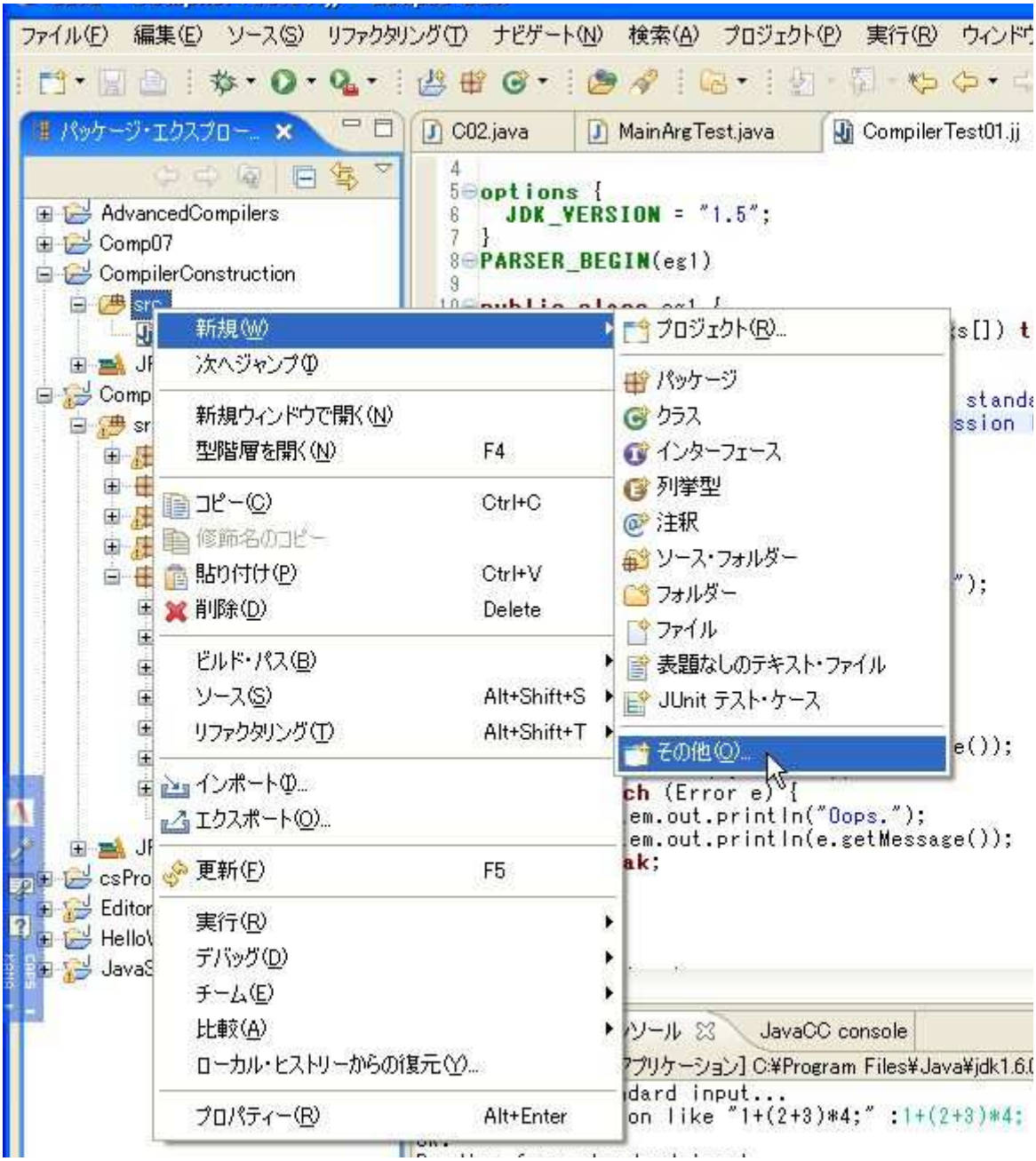
Java のバージョンを 1.5 (必須)

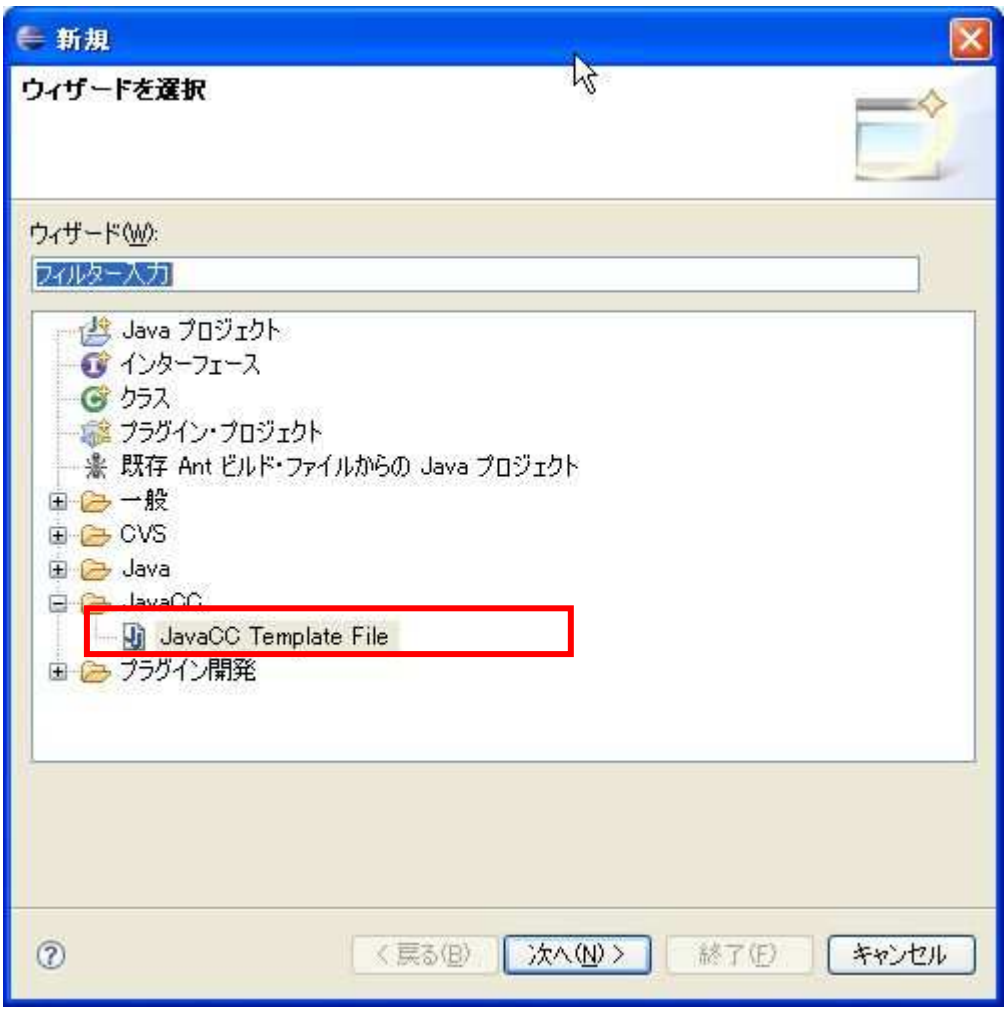


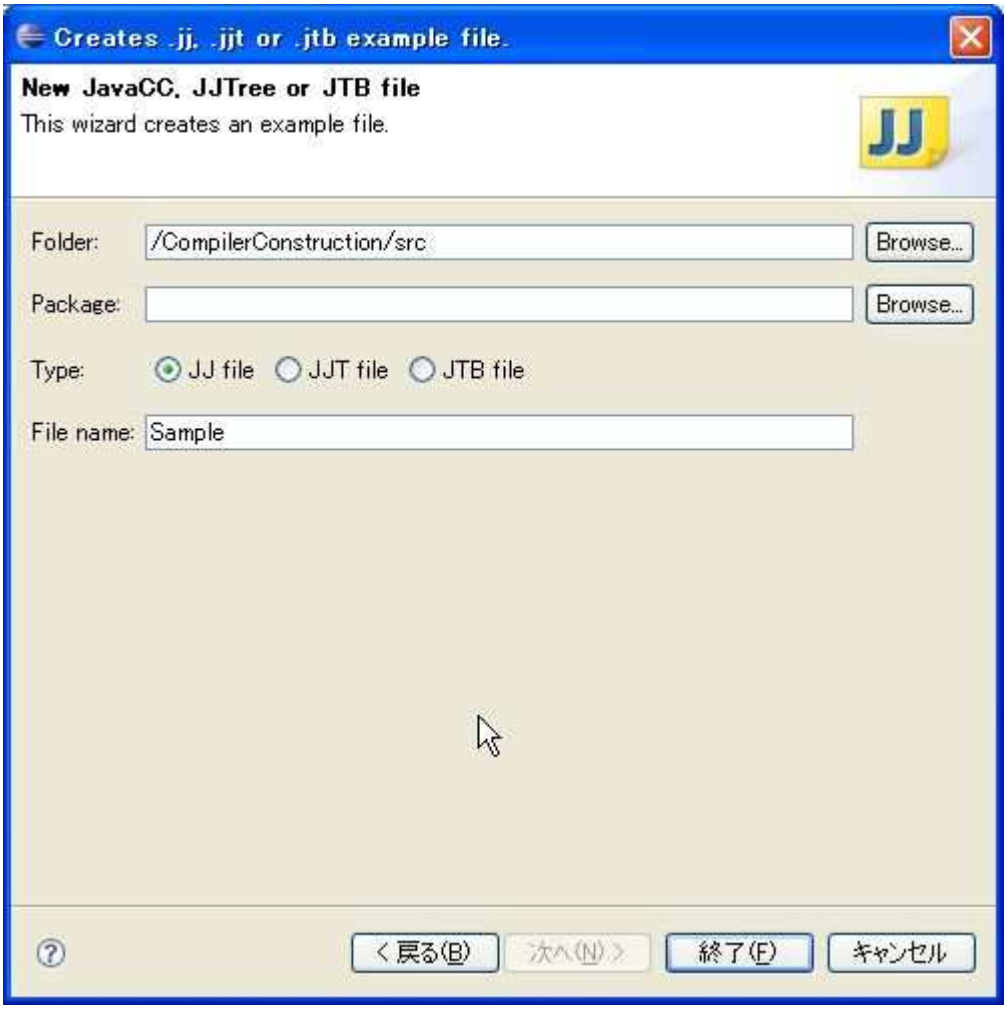
3. コンパイラ（構文解析器）の作成の流れ

準備：テンプレート作成

（ただし、ここでは実際には作成されたテンプレートは全く使わない）


ファイル(F) 編集(E) ソース(S) リファクタリング(T) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R)
パッケージ・エクスプロー...
C02.java
MainArgTest.java
CompilerTest01.jj
AdvancedCompilers
Comp07
CompilerConstruction
options {
JDK_VERSION = "1.5";
}
PARSER_BEGIN(eg1)
新規(W)
次へジャンプ(I)
新規ウィンドウで開く(N)
型階層を開く(N) F4
コピー(C) Ctrl+C
修飾名のコピー
貼り付け(P) Ctrl+V
削除(D) Delete
ビルド・パス(B)
ソース(S) Alt+Shift+S
リファクタリング(T) Alt+Shift+T
インポート(I)...
エクスポート(O)...
更新(F) F5
実行(R)
デバッグ(D)
チーム(E)
比較(A)
ローカル・ヒストリーからの復元(Y)...
プロパティー(R) Alt+Enter
プロジェクト(R)...
パッケージ
クラス
インターフェース
列挙型
注釈
ソース・フォルダー
フォルダー
ファイル
表題なしのテキスト・ファイル
JUnit テスト・ケース
その他(O)...
JavaCC console


新規
ウィザードを選択
ウィザード(W):
フィルター入力
Java プロジェクト
インターフェース
クラス
プラグイン・プロジェクト
既存 Ant ビルド・ファイルからの Java プロジェクト
一般
CVS
Java
JavaCC
JavaCC Template File
プラグイン開発
< 戻る(B)
次へ(N) >
終了(F)
キャンセル

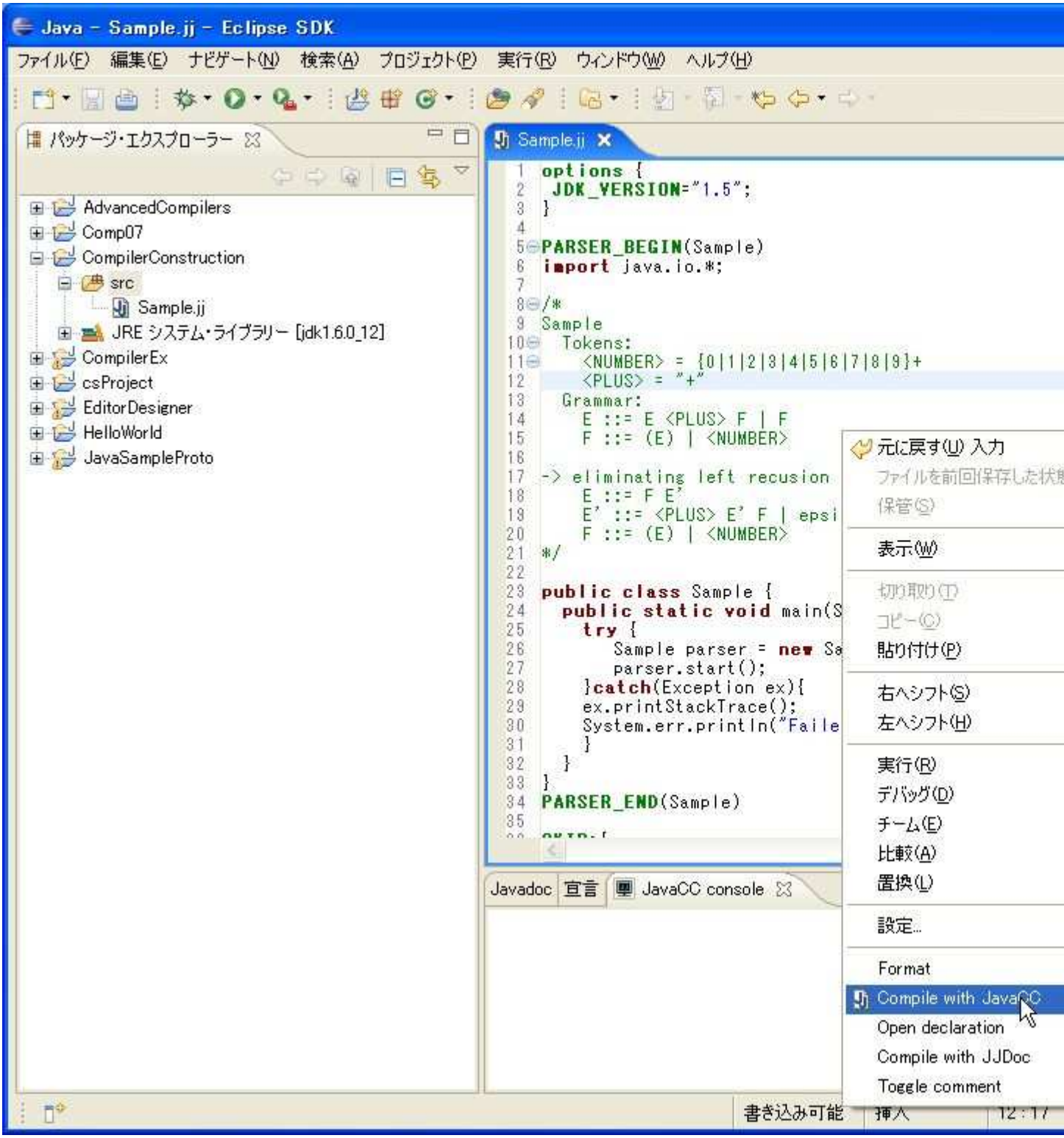


A:JJ ファイルを編集：

ここでは、テンプレートを完全に書き換えてプログラムを書く。

（本資料の「５：開発の流れ」のサンプルプログラム、あるいは配布プログラムの Sample.jj の内容で説明する。）

B:コンパイラの自動生成：　JavaCC でコンパイラ（構文解析器）を自動生成する。

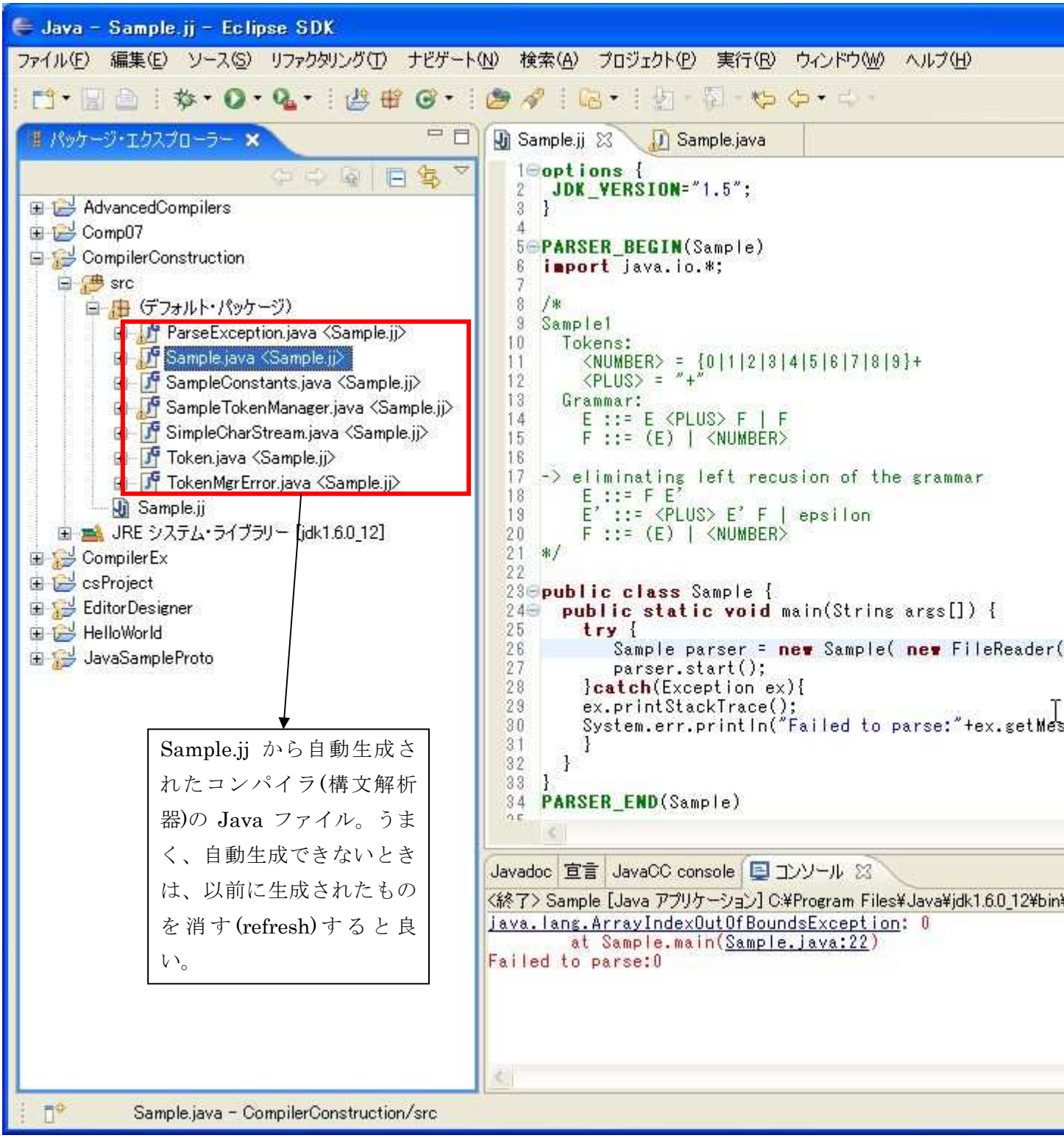


C:生成されたコンパイラ（構文解析器）のコンパイル：

実際には、Sample.jj にはバグがあるので、自動生成の段階でエラーがでるので、直さないと先には進めない。(後述)

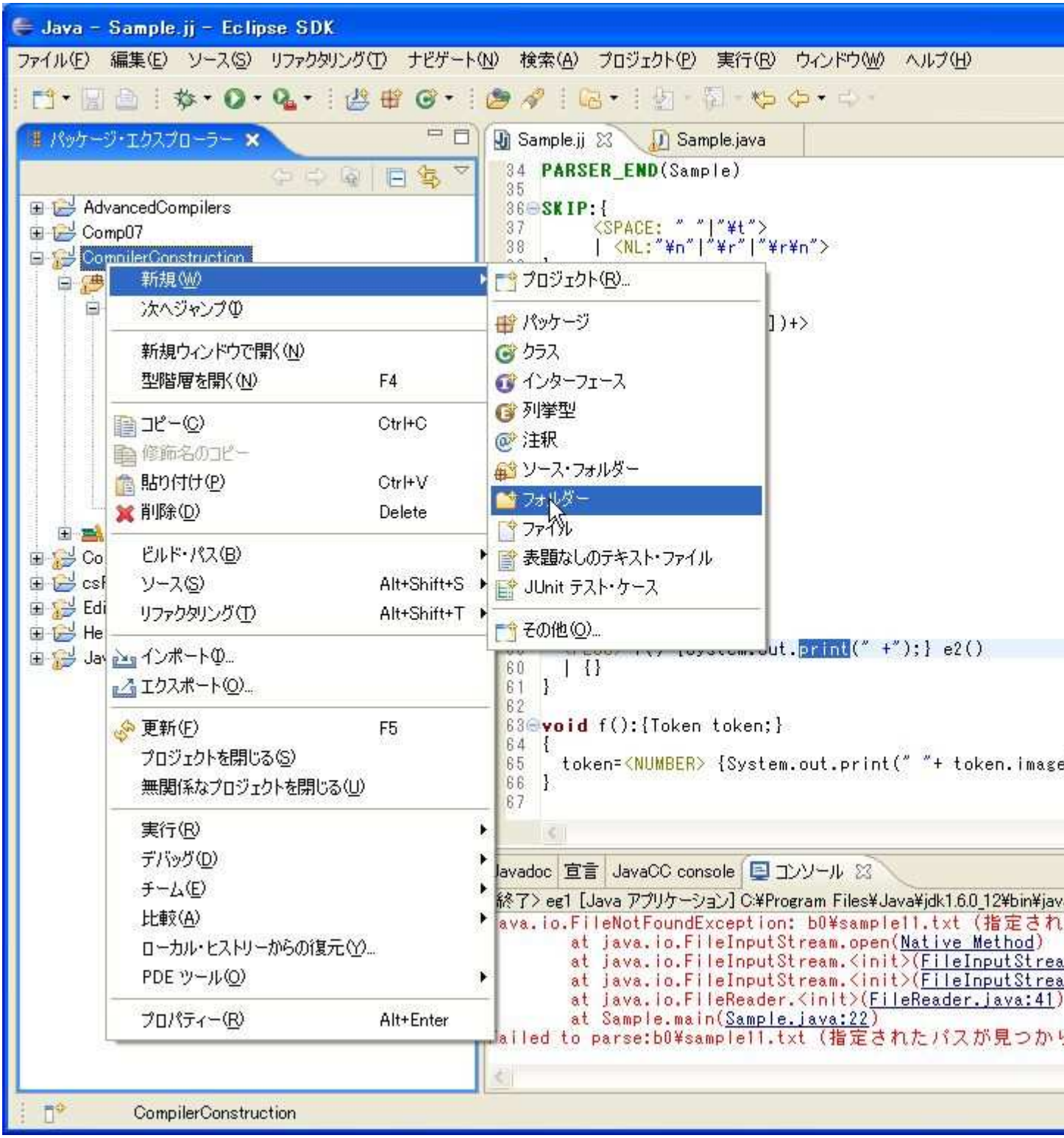
Sample.jj のバグを取り除いたあと、Sample.java（自動生成されたコンパイラ）をコンパイルする。下

記のようなエラーになればコンパイルは一応成功している。(コンパイルするべきプログラムが main の引数に与えられていないために出ているエラーである。)

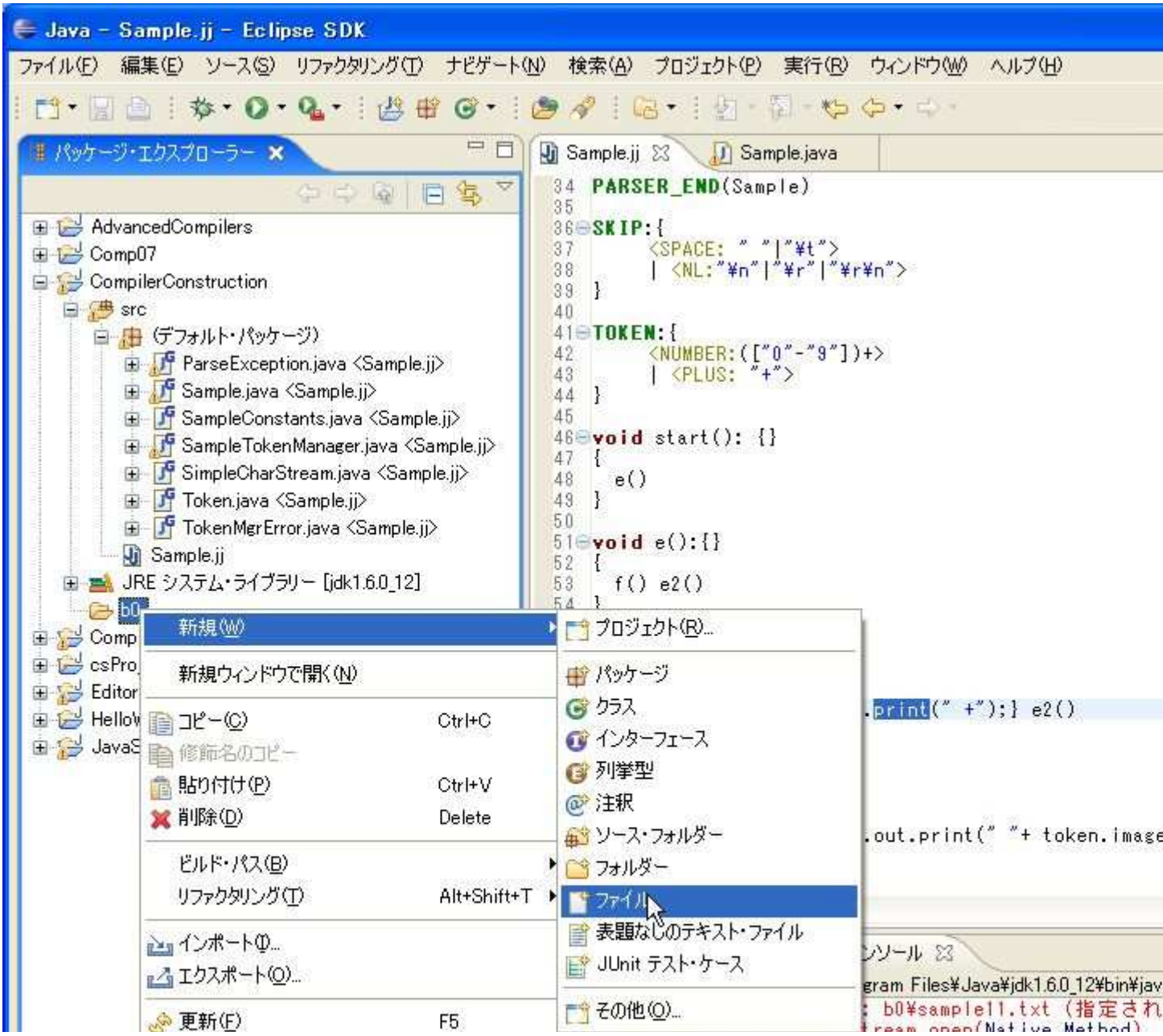


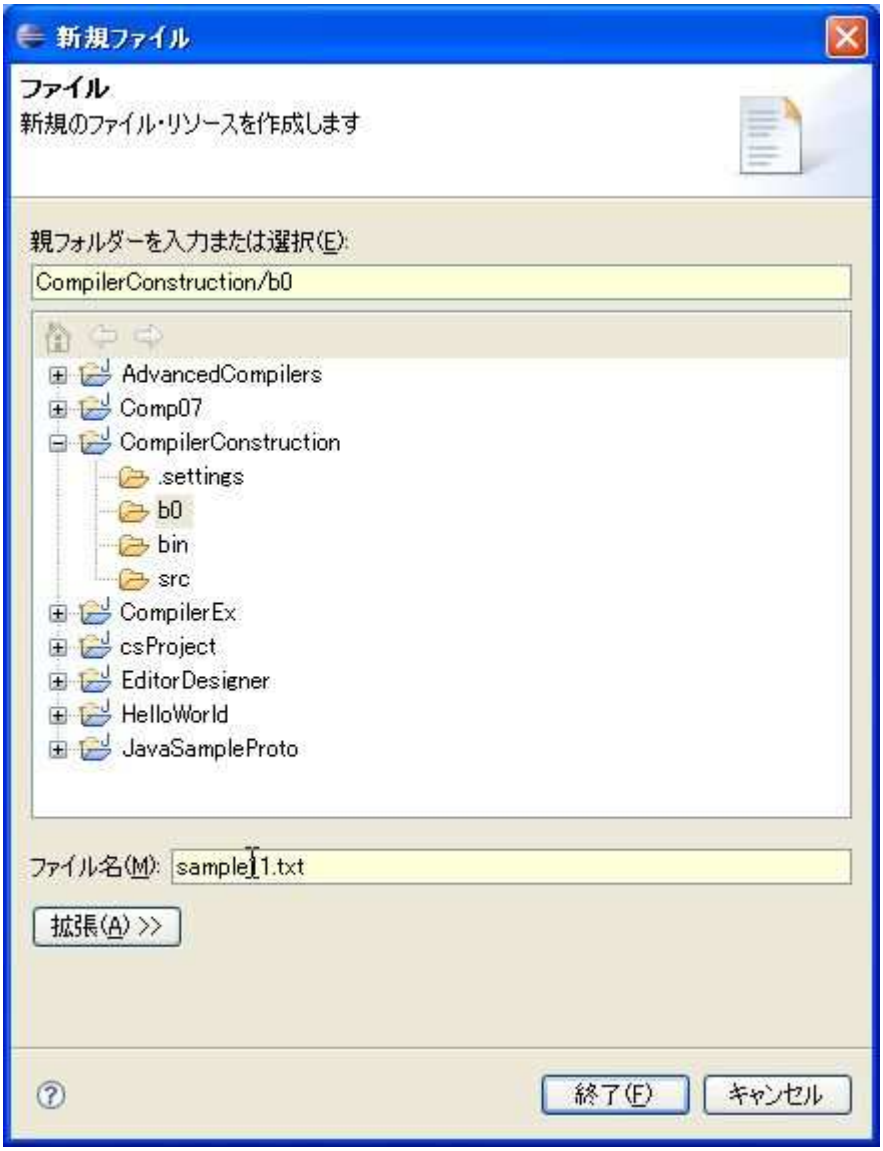
D(1):ソースコードの作成

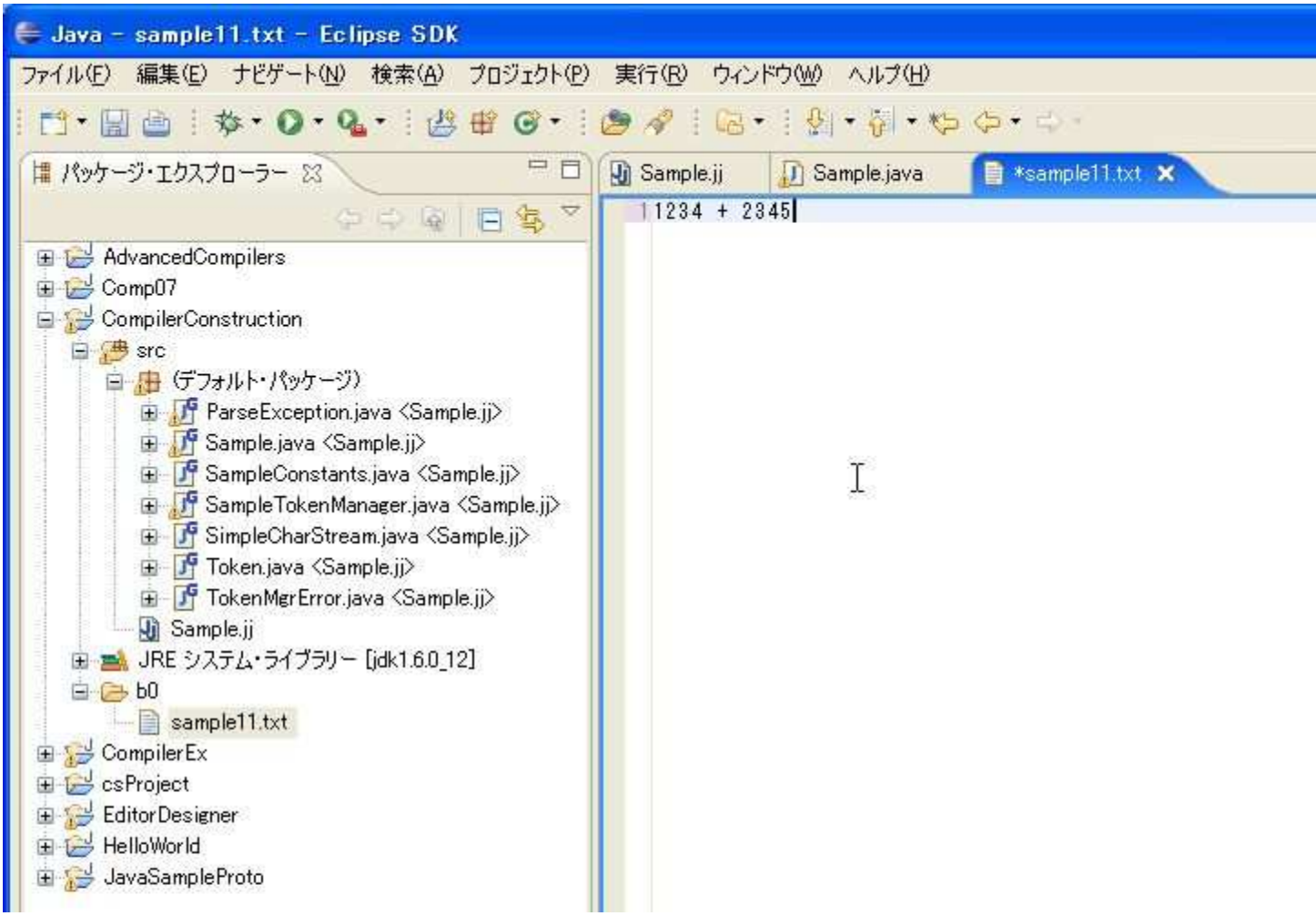
作成したコンパイラに読み込ませるソースコードを用意する。
ここでは b0 というフォルダを用意して、sample1.txt にソースコードを書く。



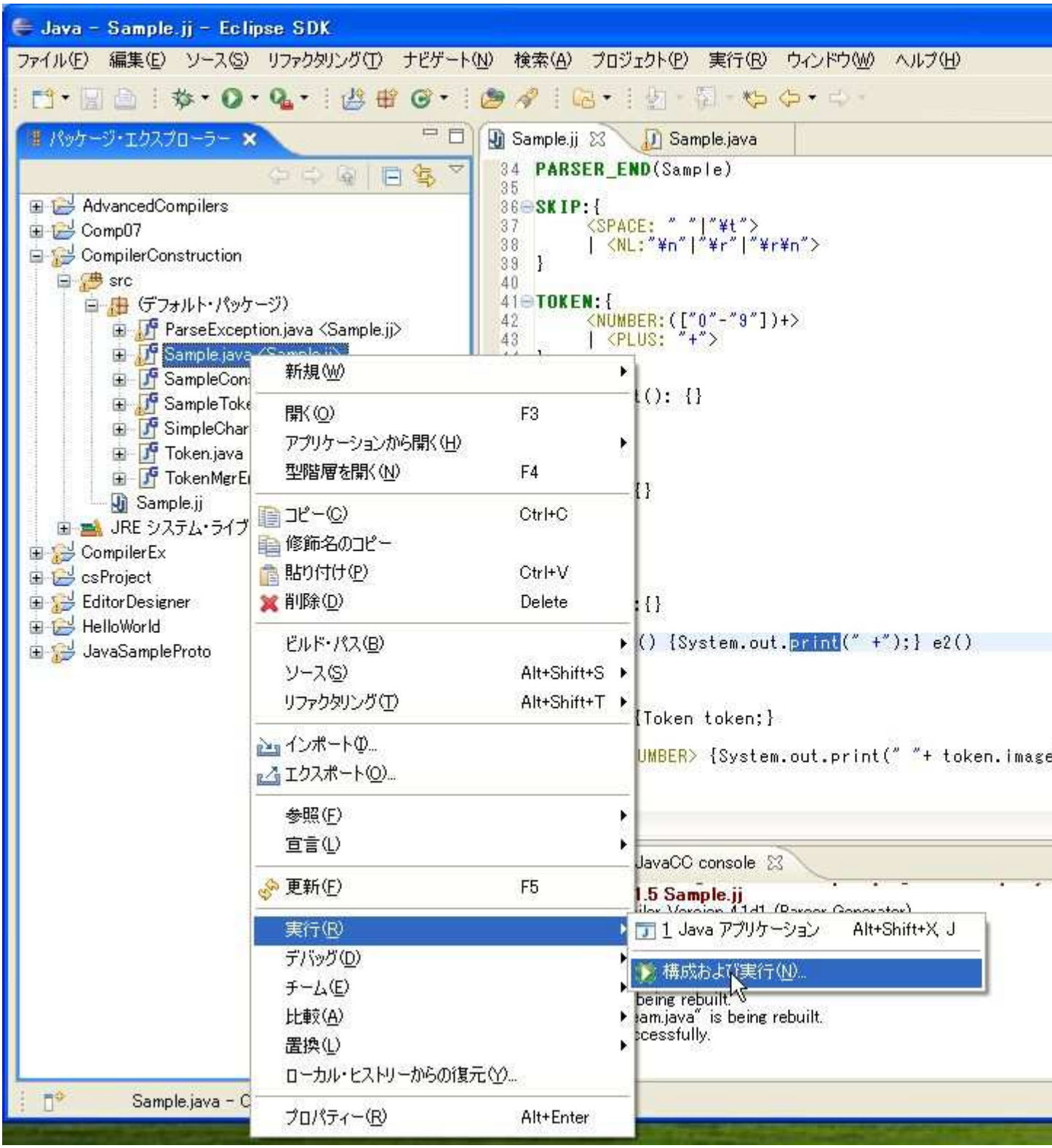

新規フォルダー
フォルダー
新規のフォルダー・リソースを作成します
親フォルダーを入力または選択(E):
CompilerConstruction
AdvancedCompilers
Comp07
CompilerConstruction
CompilerEx
csProject
EditorDesigner
HelloWorld
JavaSampleProto
フォルダー名(N): b0
拡張(A) >>
終了(F)
キャンセル

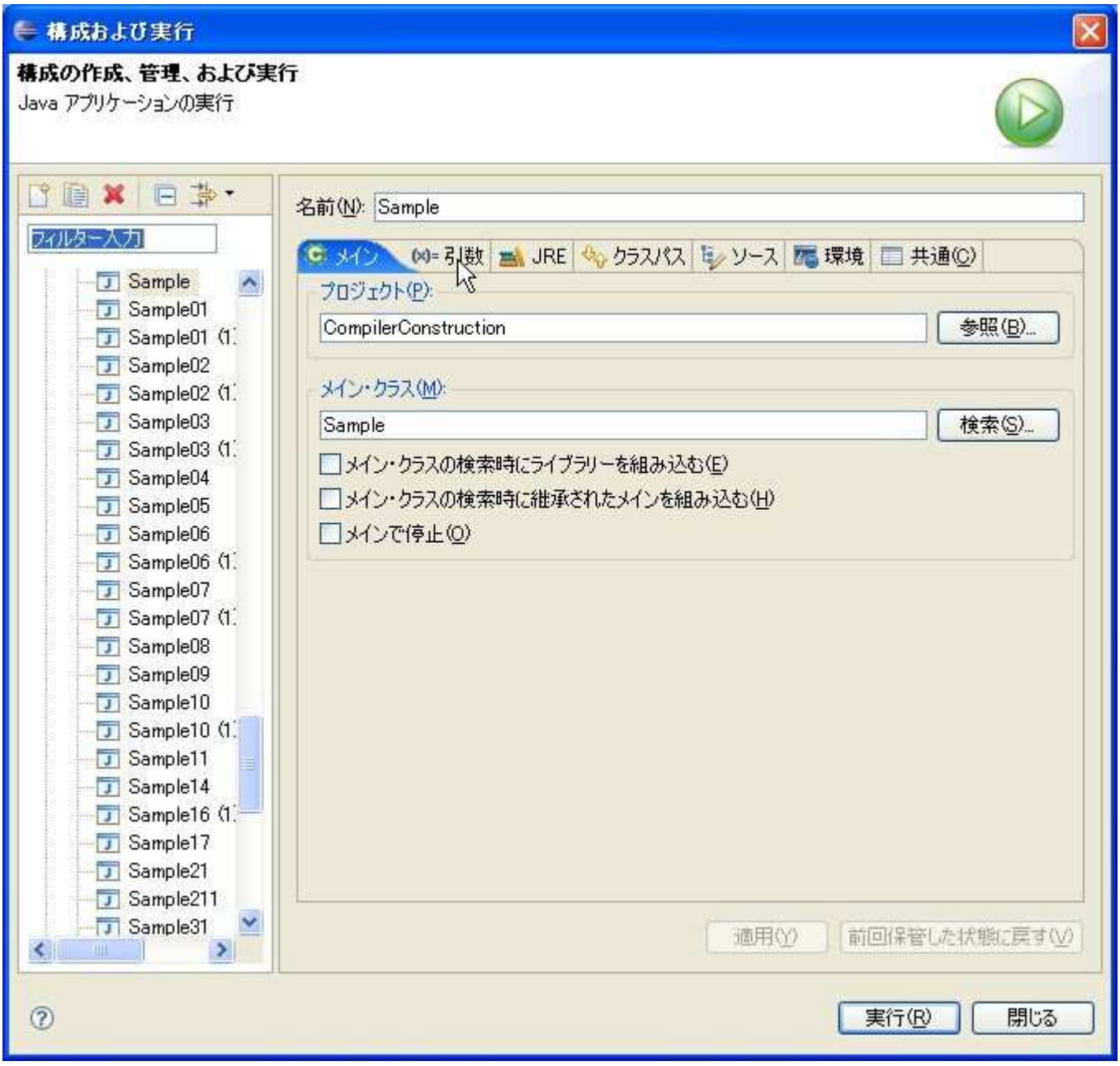

Java - Sample.jj - Eclipse SDK
ファイル(F) 編集(E) ソース(S) リファクタリング(T) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)
パッケージ・エクスプローラー
AdvancedCompilers
Comp07
CompilerConstruction
src
(デフォルト・パッケージ)
ParseException.java <Sample.jj>
Sample.java <Sample.jj>
SampleConstants.java <Sample.jj>
SampleTokenManager.java <Sample.jj>
SimpleCharStream.java <Sample.jj>
Token.java <Sample.jj>
TokenMgrError.java <Sample.jj>
Sample.jj
JRE システム・ライブラリー [jdk1.6.0_12]
Sample.jj
Sample.java
PARSER_END(Sample)
SKIP:{
<SPACE: " "|"¥t">
| <NL:"¥n"|"¥r"|"¥r¥n">
}
TOKEN:{
<NUMBER:(["0"-"9"])+>
| <PLUS: "+">
}
void start(): {}
{
e()
}
void e():{}
{
f() e2()
新規(W)
新規ウィンドウで開く(N)
コピー(C) Ctrl+C
修飾名のコピー
貼り付け(P) Ctrl+V
削除(D) Delete
ビルド・パス(B)
リファクタリング(T) Alt+Shift+T
インポート(I)...
エクスポート(O)...
更新(F) F5
プロジェクト(R)...
パッケージ
クラス
インターフェース
列挙型
注釈
ソース・フォルダー
フォルダー
ファイル
表題なしのテキスト・ファイル
JUnit テスト・ケース
その他(O)...


新規ファイル
ファイル
新規のファイル・リソースを作成します
親フォルダーを入力または選択(E):
CompilerConstruction/b0
AdvancedCompilers
Comp07
CompilerConstruction
.settings
b0
bin
src
CompilerEx
csProject
EditorDesigner
HelloWorld
JavaSampleProto
ファイル名(M): sample1.txt
拡張(A) >>
終了(F)
キャンセル

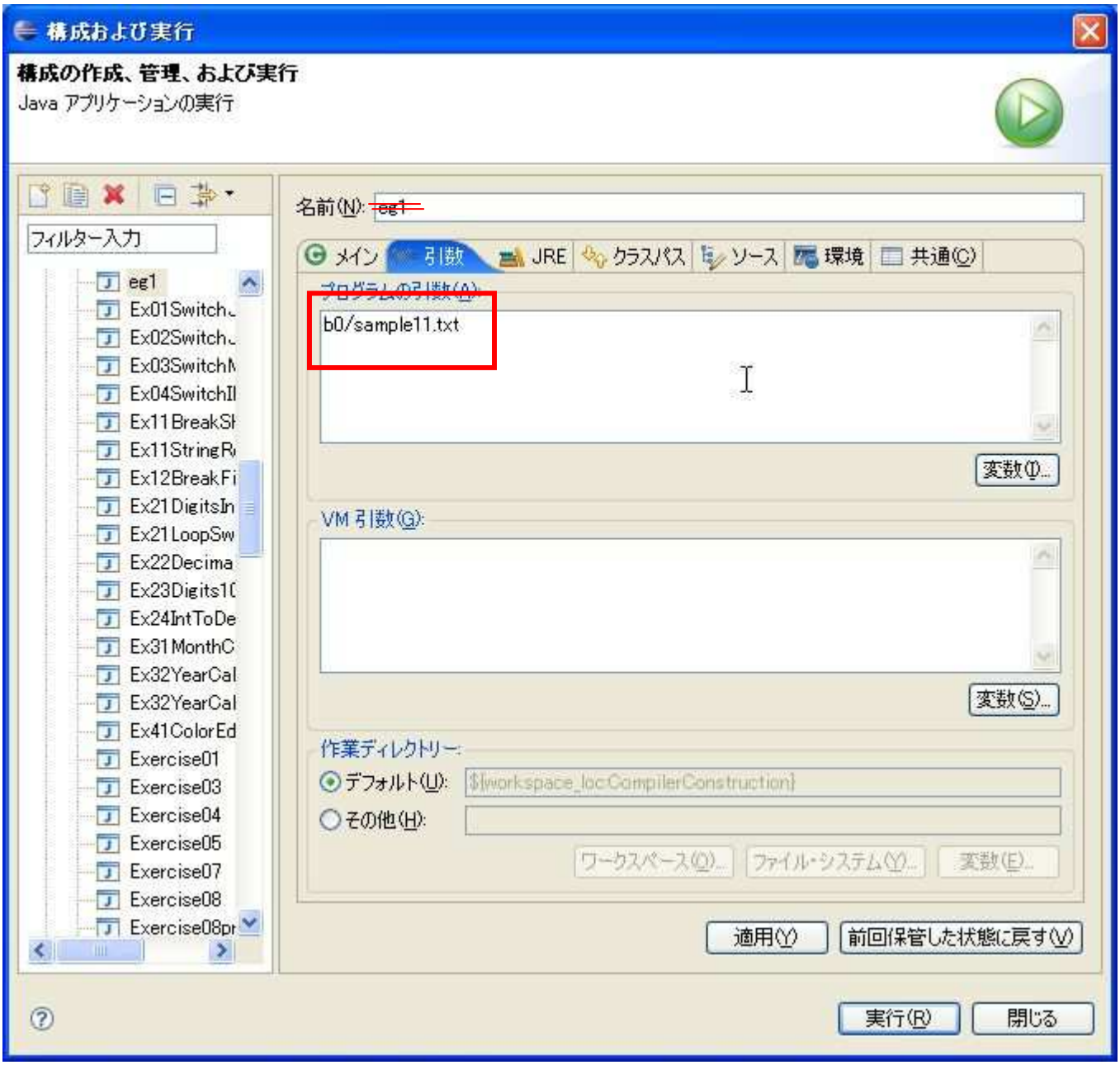


D(2) 作成したコンパイラを実際に動かす。


Java - Sample.jj - Eclipse SDK
ファイル(F) 編集(E) ソース(S) リファクタリング(T) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)
パッケージ・エクスプローラー
AdvancedCompilers
Comp07
CompilerConstruction
src
(デフォルト・パッケージ)
ParseException.java <Sample.jj>
Sample.java
Sample.jj
JRE システム・ライブ
CompilerEx
csProject
EditorDesigner
HelloWorld
JavaSampleProto
Sample.jj
Sample.java
PARSER_END(Sample)
SKIP:{
<SPACE: " "|"¥t">
| <NL:"¥n"|"¥r"|"¥r¥n">
}
TOKEN:{
<NUMBER:(["0"-"9"])+>
| <PLUS: "+">
新規(W)
開く(O) F3
アプリケーションから開く(H)
型階層を開く(N) F4
コピー(C) Ctrl+C
修飾名のコピー
貼り付け(P) Ctrl+V
削除(D) Delete
ビルド・パス(B)
ソース(S) Alt+Shift+S
リファクタリング(T) Alt+Shift+T
インポート(I)...
エクスポート(O)...
参照(F)
宣言(L)
更新(F) F5
実行(R)
デバッグ(D)
チーム(E)
比較(A)
置換(L)
ローカル・ヒストリーからの復元(Y)...
プロパティー(R) Alt+Enter
1 Java アプリケーション Alt+Shift+X, J
構成および実行(N)...
JavaCC console
1.5 Sample.jj
being rebuilt.
Sample.java - C



引数にコンパイルしたいファイル名を書く。（さきほどの 1234 +2345 をコンパイルする。）



コンソールに下記が表示されたら成功。

```
1234 2345 +
```

4. コンパイラ作成の流れ

A. jj ファイルを編集、作成。（上の例では Sample.jj）

B. コンパイラの生成　（jj ファイルに誤りがあると、先には進まないので**Aにもどる**。）

　成功すると Sample.java が作られる。

C. Sample.java のコンパイルと実行

Sample.java のコンパイルに成功すると、コンパイラが起動する。

ただし、Sample.jj のアクションの部分などに誤りがあった場合、Sample.java をコンパイルすることはできないので、**A に戻る。**

D. 実際にコンパイラの動きを試す。動きがおかしいときは **A に戻る。**

5. 開発の例：

□version 1

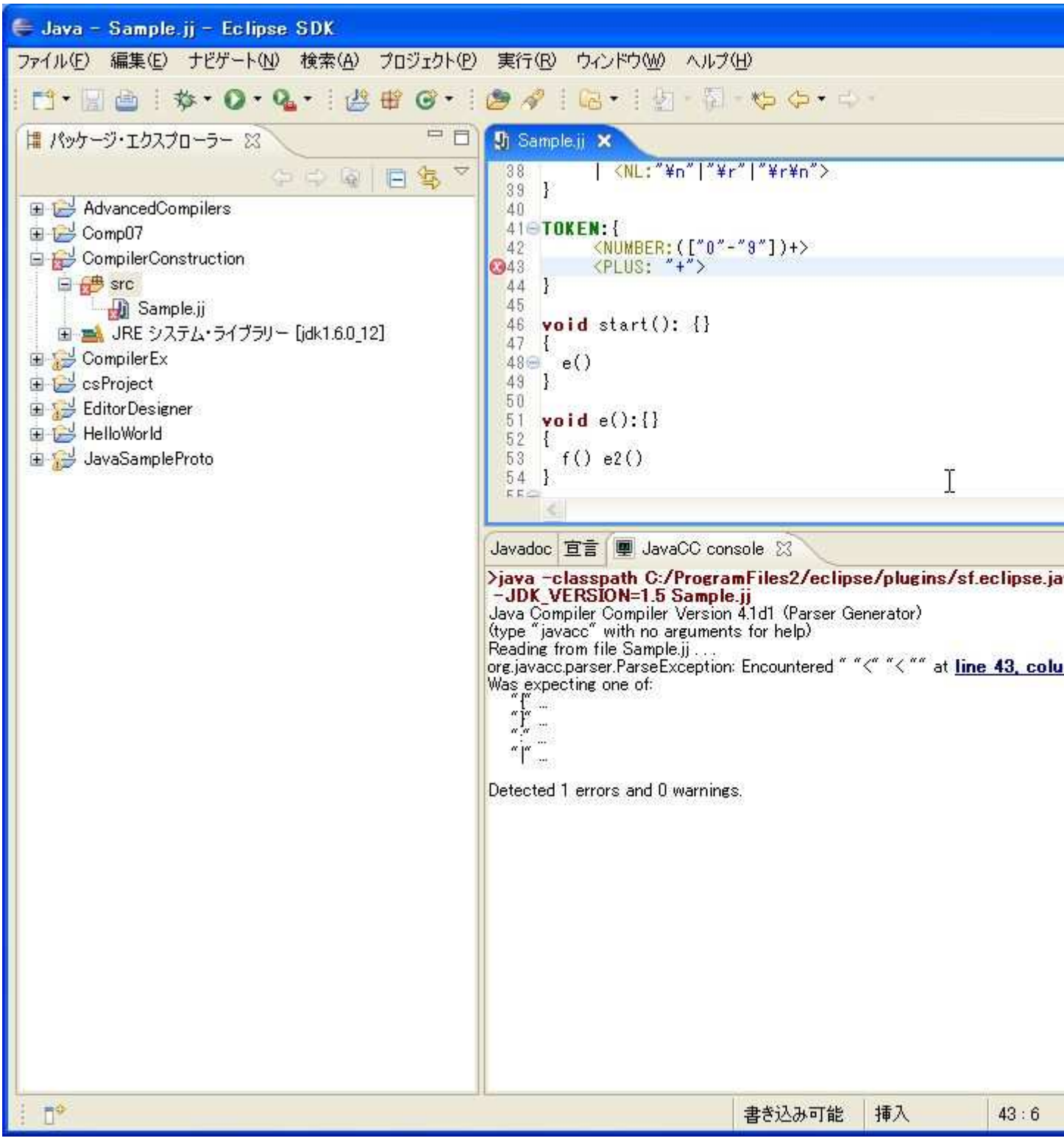
○4A. サンプルプログラム Sample.jj（わざと誤りが入れてある）の作成

```
options {
  JDK_VERSION="1.5";
}

PARSER_BEGIN(Sample)
import java.io.*;
/*
Sample
   Tokens:
      <NUMBER> = {0|1|2|3|4|5|6|7|8|9}+
      <PLUS> = "+"
   Grammar:
      E ::= E <PLUS> F | F
      F ::= (E) | <NUMBER>

-> eliminating left recursion of the grammar
      E ::= F E'
      E' ::= <PLUS> E' F | epsilon
      F ::= (E) | <NUMBER>
*/

public class Sample {
   public static void main(String args[]) {
      try {
          Sample parser = new Sample( new FileReader(args[0] ));
          parser.start();
      }catch(Exception ex){
      ex.printStackTrace();
      System.err.println("Failed to parse:"+ex.getMessage());
```

```
    }
  }
}
PARSER_END(Sample)

SKIP:{
      <SPACE: " "|"¥t">
      | <NL:"¥n"|"¥r"|"¥r¥n">
}

TOKEN:{
      <NUMBER:(["0"-"9"])+>
      <PLUS: "+">
}

void start(): {}
{
  e()
}

void e():{}
{
  f() e2()
}

void e2():{}
{
  <PLUS> f() {System.out.pritn(" +");} e2()   /*  誤り  */
  | {}
}

void f():{Token token;}
{
  token=<NUMBER> {System.out.print(" "+ token.image);}
}
```

○4B.　コンパイラの自動生成

→jj ファイルの記述誤りがあるのでエラーがでる。

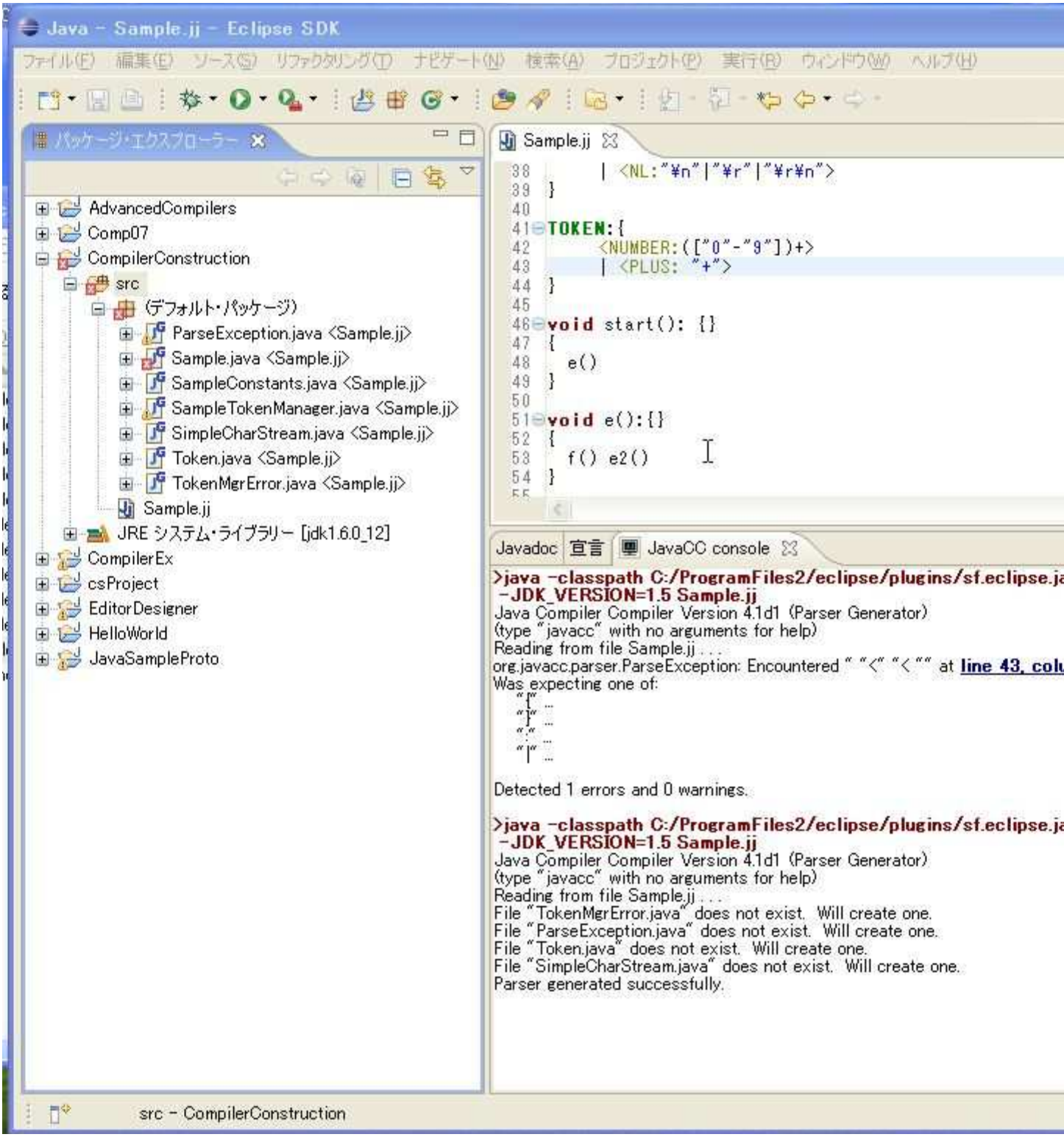


☐version 2

○4A→jj ファイルを修正し Version 2 を作る。

「｜」を加えると直る。上記 43 行目の　<Plus>の前。

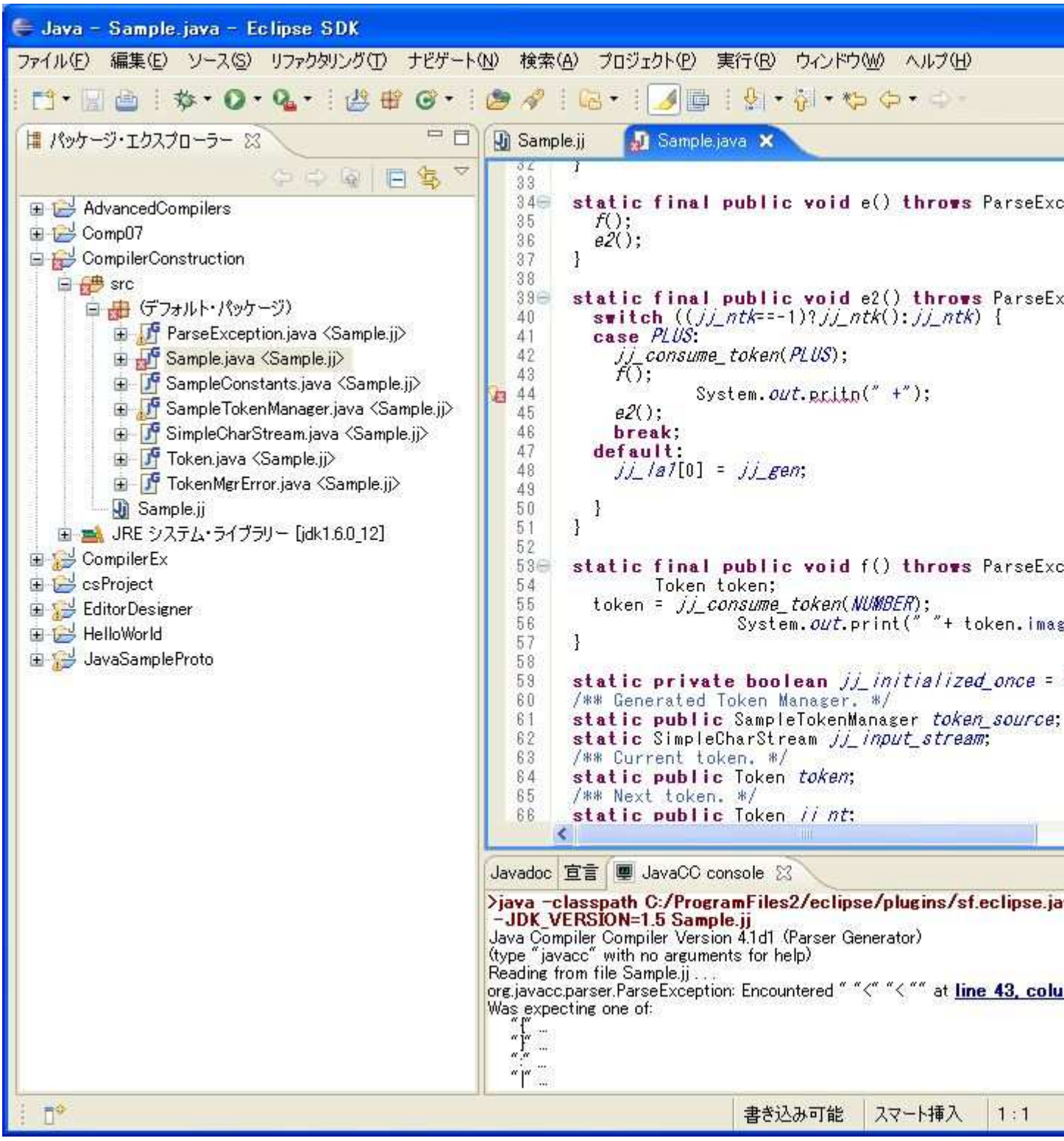
○4B:コンパイラの生成→成功



4C　生成されたコンパイラのコンパイルと実行

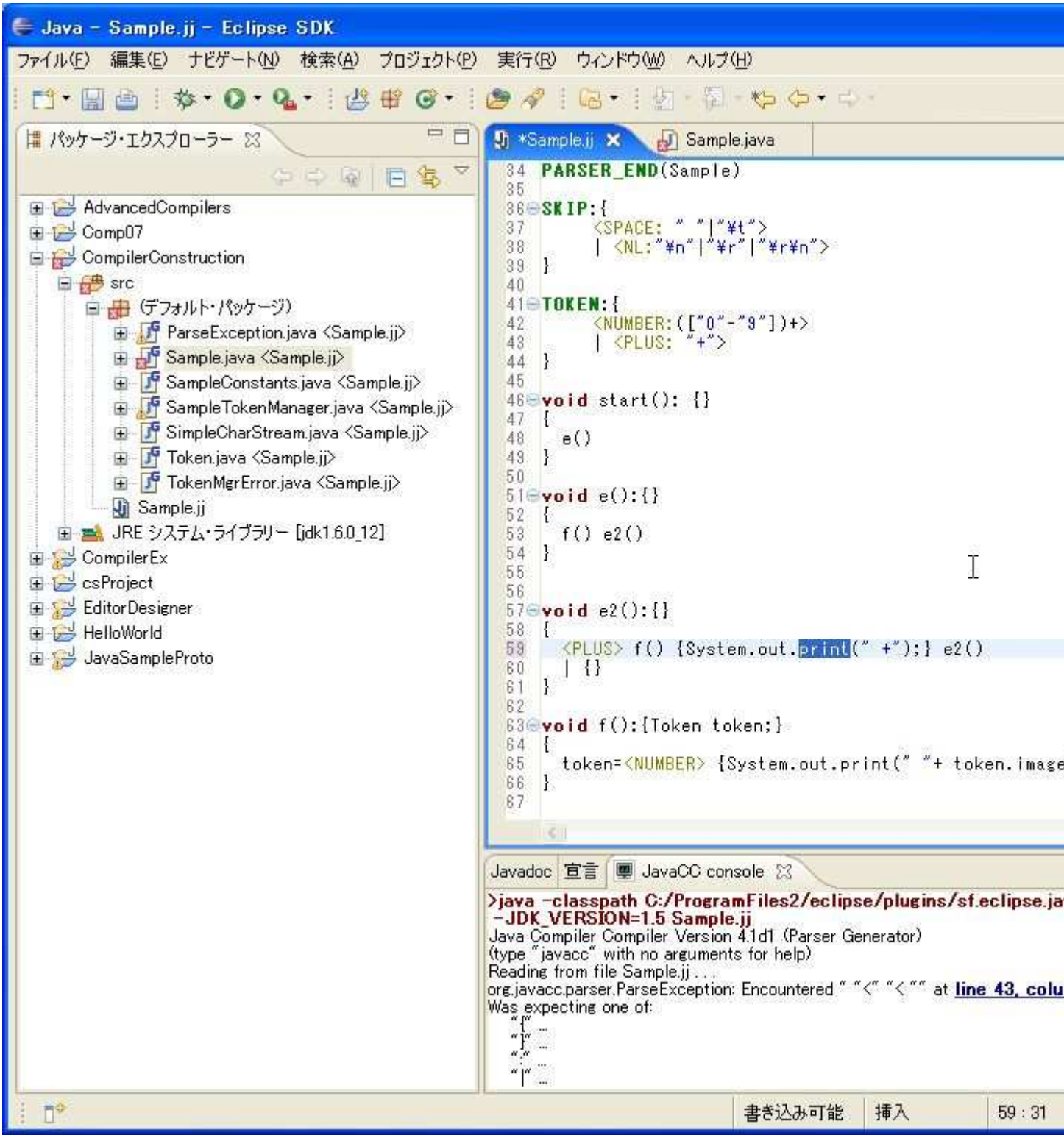
→ Sample.java に赤×印がついているが、生成された Java ファイルに誤りがある。print のつづりあや

まり。もとの jj ファイルを直す必要がある。



□version 3

○4A→jj ファイルを修正



○4B: コンパイラの自動生成→成功

→成功

○4C、4D　生成されたコンパイラのコンパイルと実行

「３.コンパイラ（構文解析器）の作成の流れ」のＤの項に書いた方法で、ソースプログラムを用意した後、実際に作成したコンパイラを動かしてみる。

サンプルテキスト、sample1.txt

```
1234+2345
```

実行結果(コンソールに出力される)

```
1234　2345　+
```

6. 配布したサンプル

配布したサンプルに含まれる javaccsamples 以下の jj ファイルおよび txt ファイルの説明

□字句解析のテスト→ScannerSample1.jj

このテストは、標準入力(System.in)から入力するので、実行する際には引数は必要なし。

実行するとコンソールで入力待ちになるので、テキストを入力する。(Enter を押さないと入力は処理されない。)

□算術式を逆ポーランドに翻訳するプログラム

- Sample.jj 上述のエラーのある例。修正したものが Sample1.jj
- Sample1.jj 足し算その 1
  サンプルソースプログラムは、sample11.txt, sample12.txt
- Sample2.jj 足し算その 2
  サンプルソースプログラムは、sample11.txt, sample12.txt
- Sample3.jj 括弧を扱えるようにする。
  サンプルソースプログラムは、sample11.txt, sample31.txt
- Sample4.jj 引き算を扱えるようにする。
  サンプルソースプログラムは、sample11.txt, sample41.txt
- Sample5.jj 掛け算を扱えるようにする。
  サンプルソースプログラムは、sample11.txt, sample51.txt