
Разработка объектно-ориентированных систем программирования интегрированных в среду Eclipse

2. Разработка распознавателей объектно-ориентированных языков программирования

Владимир Юрьевич Романов,
Московский Государственный Университет им. М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
vromanov@cmc.msu.ru,
romanov.rvy@yandex.ru

Генератор компиляторов ANTLR 4

ANother Tool for Language
Recognition

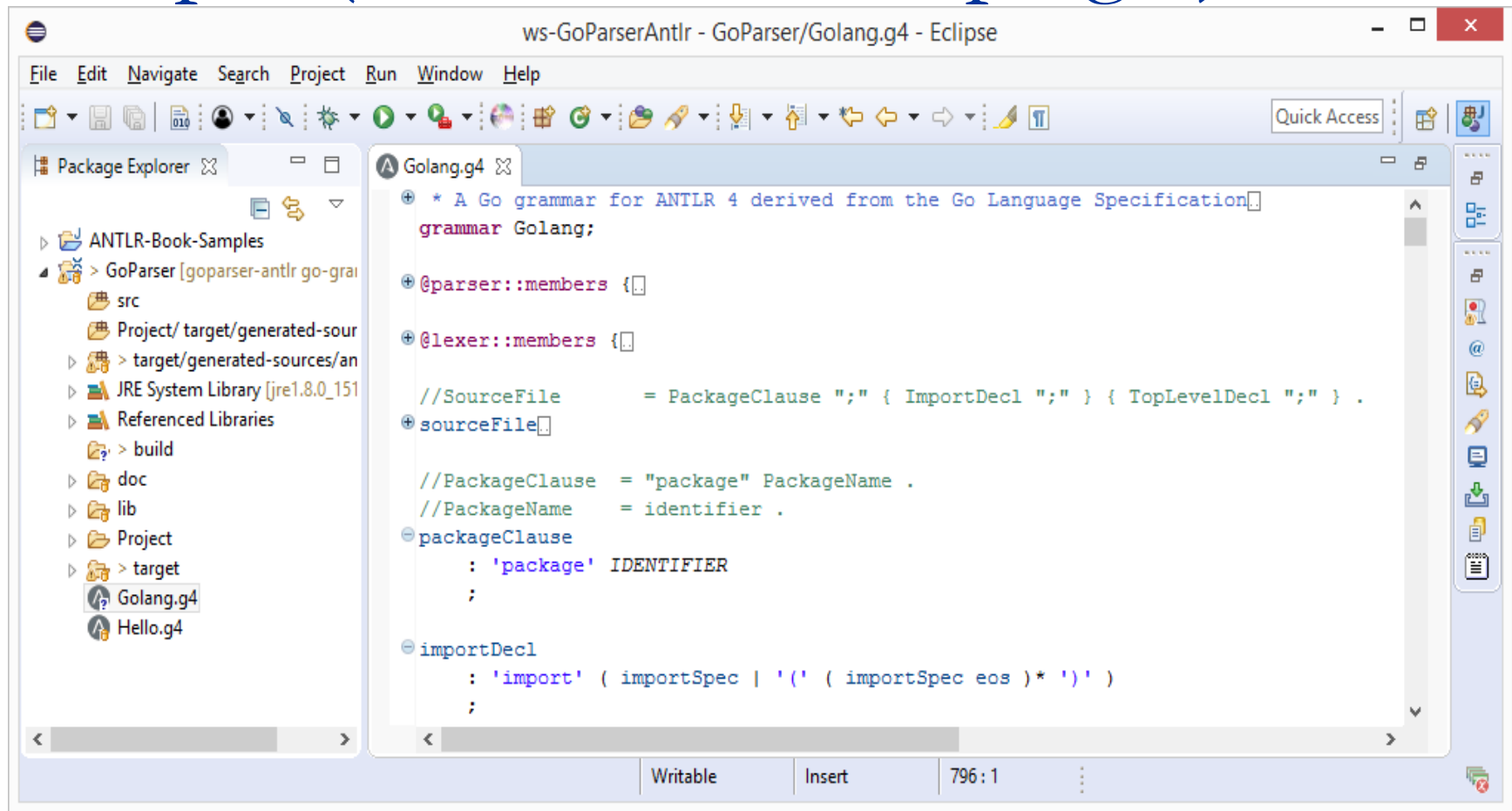
Установка ANTLR 4 в среде Eclipse

- ANTLR – описание инструмента
<http://www.antlr.org/>
- Download ANTLR
<https://www.antlr.org/download.html>
 - ANTLR tool itself
<https://www.antlr.org/download/antlr-4.9.3-complete.jar>
- Расширение среды Eclipse для ANTLR 4.
<https://github.com/jknack/antlr4ide#installation>
- Грамматики языков описанные на ANTLR 4
<https://github.com/antlr/grammars-v4>

Грамматики описанные на языке ANTLR

abnf agc antlr3 **antlr4** apex asm6502 asn aspectj atl
basic bnf brainfuck c calculator capnproto clf cliff
clojure cobol85 cookie cool cpp creole csharp css3 csv
datetime dcm dice dot ecmascript edif300 emailaddress
erlang fasta fen fol fortran77 gml **golang** graphql
graphstream-dgs gtin html hypertalk icalendar idl inf
informix iri istc **java java8 java9** javadoc javascript
jpa json lambda logo lua masm modelica modula2pim4 norma
lize molecule mps umath mumps muparser mysql objc oncrpc
pascal pcre pddl pdn pdp7 peoplecode pgn php plsql prolog
propcalc properties protobuf3 python2-js python2 python3-
js python3-py python3-ts python3 python3alt quakemap r
rcs redcode restructuredtext rexx robotwars ruby scala
scss sgf sharc smalltalk smiles smlibv2 snobol solidity
sparql sqlite stacktrace stringtemplate suokif swift-fin
swift2 swift3 thrift tiny tinybasic tinyc tnsnames tnt
tsql turtle turtle ucb-logo unicode upnp url vb6 vba
verilog vhdl wavefront webidl xml xpath z

Грамматика языка GoLang в среде Eclipse (ANTLR 4 IDE plugin)



```
* A Go grammar for ANTLR 4 derived from the Go Language Specification
grammar Golang;

@parser::members {
}

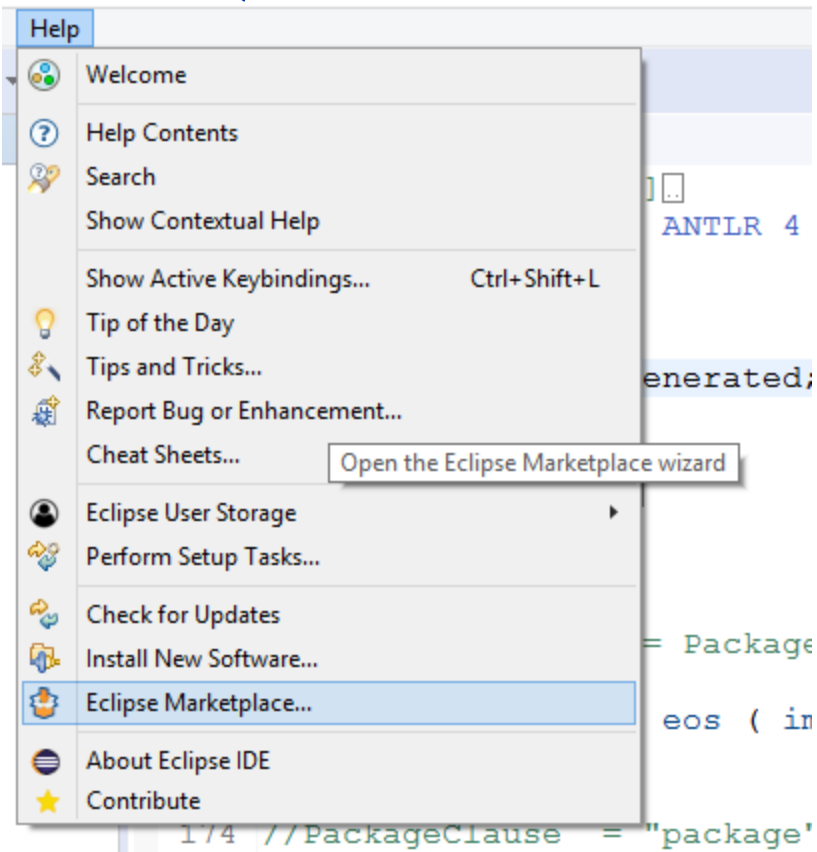
@lexer::members {
}

//SourceFile      = PackageClause ";" { ImportDecl ";" } { TopLevelDecl ";" } .
sourceFile

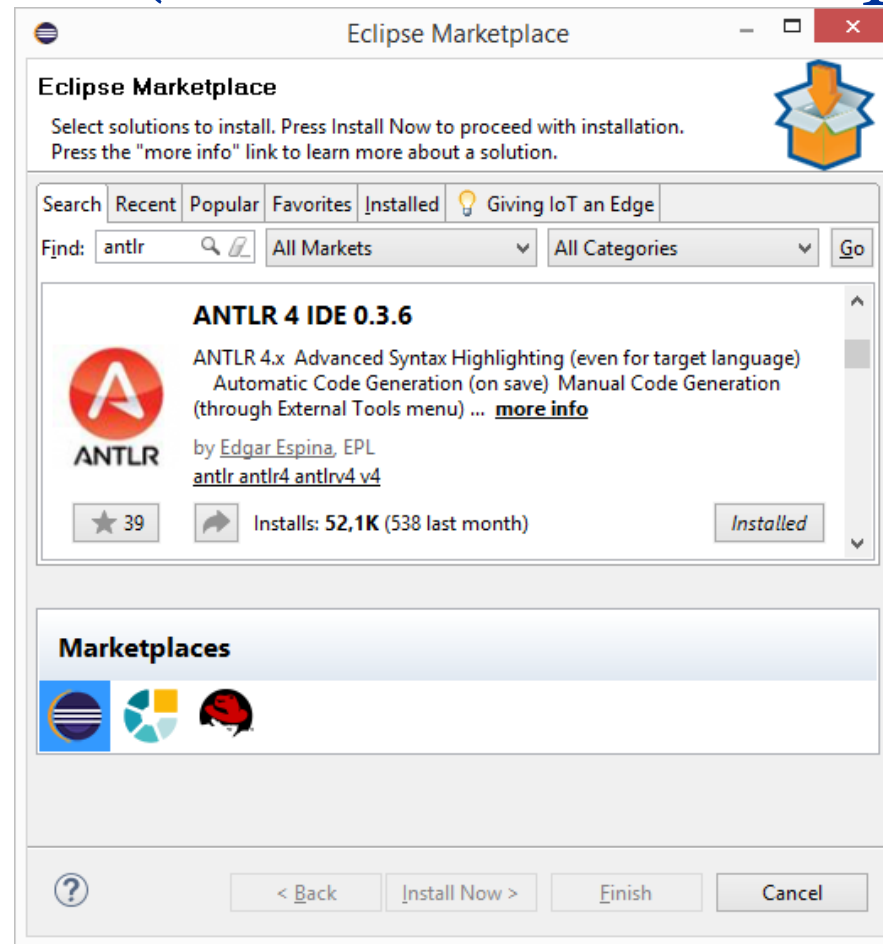
//PackageClause  = "package" PackageName .
//PackageName    = identifier .
packageClause
    : 'package' IDENTIFIER
    ;

importDecl
    : 'import' ( importSpec | '(' ( importSpec eos )* ')' )
    ;
```

Установка плагина с Eclipse Marketplace (ANTLR 4 IDE plugin)



Установка плагина с Eclipse Marketplace (ANTLR 4 IDE plugin)



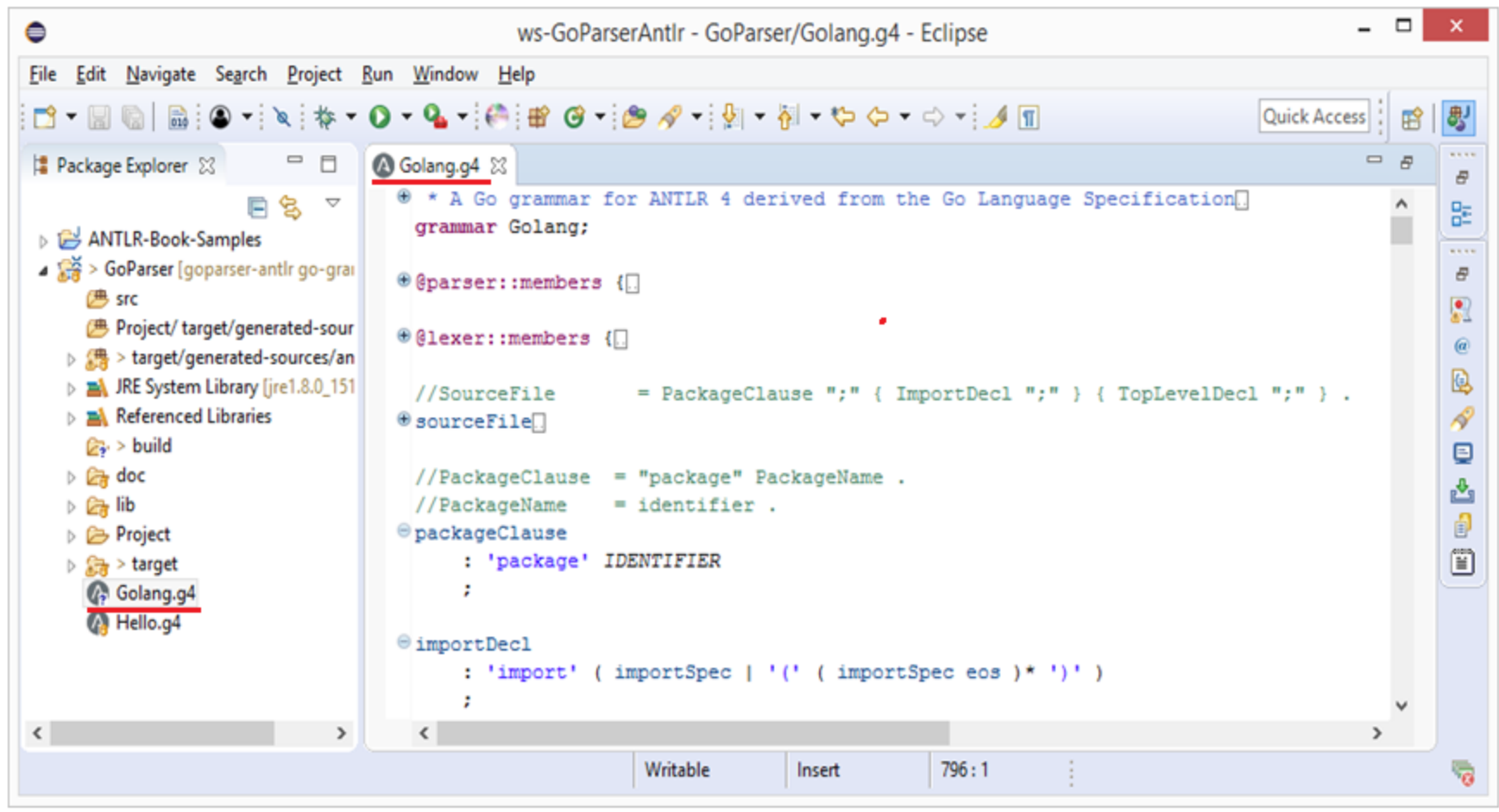
Установка плагина с Eclipse Marketplace (ANTLR 4 IDE plugin)

<https://marketplace.eclipse.org/content/antlr-4-ide>

The screenshot shows the Eclipse Marketplace interface for the ANTLR 4 IDE plugin. The page layout includes a top navigation bar with the Eclipse Marketplace logo and links for 'My Marketplace', 'Add Content', and 'More'. Below the navigation bar is a breadcrumb trail: 'Home / Marketplace / Tools (1605) / ANTLR 4 IDE'. The main content area is divided into several sections:

- MARKETS**: A section with a right-pointing arrow.
- SEARCH**: A search box with a refresh icon, an 'Advanced Search' dropdown, and a 'Search' button.
- MORE LIKE THIS**: A list of related plugins: XVisitorDT, ANTLR IDE, AntlrDT, DSL Forge, and Eclipse Xtext.
- ★ FAVORITED BY**: A section with a right-pointing arrow.
- ANTLR 4 IDE**: The main plugin details section, featuring the ANTLR logo, a star rating of 39, a comment count of 1, and an 'Install' button. Below the button are icons for home, heart, question, and download.
- Details**: A tabbed interface with sub-tabs for 'Screenshots', 'Metrics', 'Errors', and 'External Install Button'. The 'Details' tab is active, showing a list of features:
 - ANTLR 4.x
 - Advanced Syntax Highlighting (even for target language)
 - Automatic Code Generation (on save)
 - Manual Code Generation (through External Tools menu)
 - Code Formatter (Ctrl+Shift+F)
 - Syntax Diagrams as HTML
 - Live Parse Tree evaluation
 - Advanced Rule Navigation between files (F3 or Ctrl+Click over a rule)
 - Quick fixes
- Categories**: Editor, IDE, Programming Languages
- Tags**: antlr antlr4 antlrv4 v4
- Additional Details**: A section with a dropdown arrow containing:
 - Eclipse Versions**: 2019-03 (4.11), 2018-12 (4.10), 2018-09 (4.9), Photon (4.8), Oxygen (4.7), Neon (4.6), Mars (4.5), Luna (4.4), Kepler (4.3), Juno (4.2, 3.8), Previous to Juno (<=4.1), 2019-06 (4.12), 2019-09 (4.13), 2019-12 (4.14), 2020-03 (4.15)
 - Platform Support**: Windows, Mac, Linux/GTK
 - Organization Name**: Edgar Espina
 - Date Created**: Tue, 2014-12-09 09:04
 - Development Status**: Beta
 - License**: EPL
 - Date Updated**: Fri, 2020-01-24 10:16
 - Submitted by**: Edgar Espina

Грамматика языка GoLang в среде Eclipse (ANTLR 4 IDE plugin)



```
* A Go grammar for ANTLR 4 derived from the Go Language Specification
grammar Golang;

@parser::members {

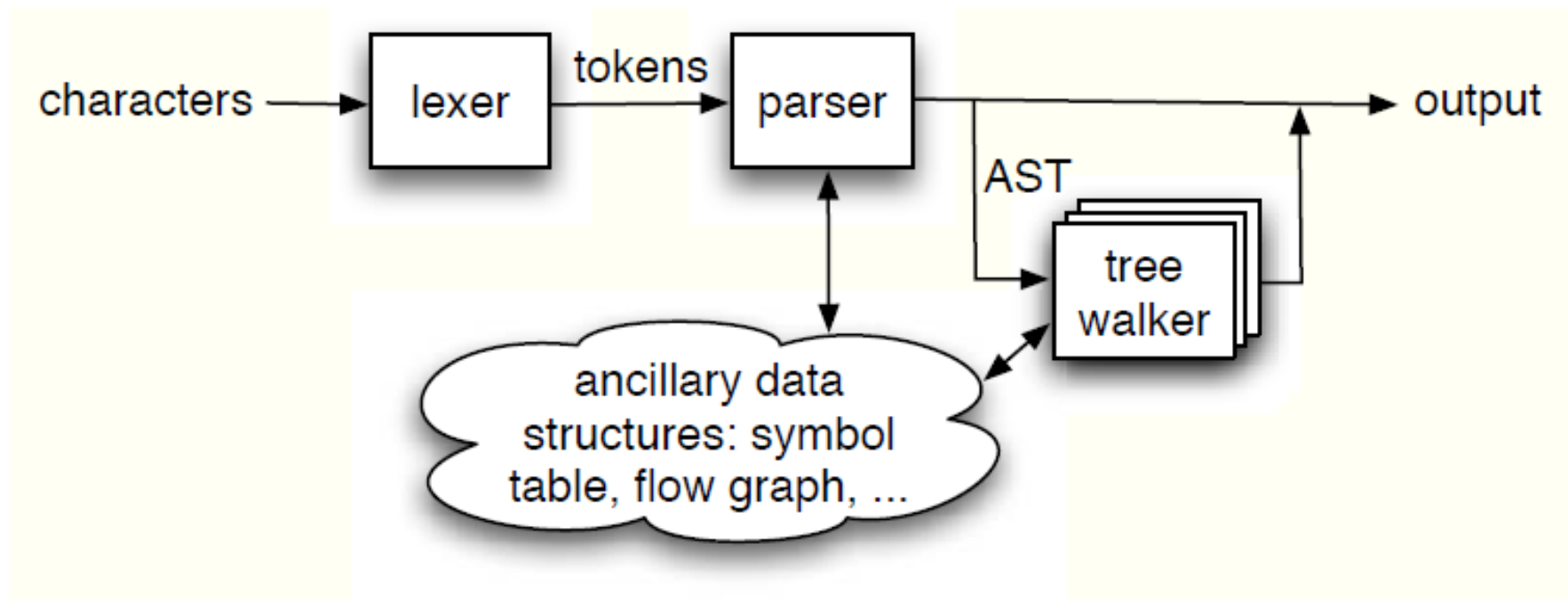
@lexer::members {

//SourceFile      = PackageClause ";" { ImportDecl ";" } { TopLevelDecl ";" } .
sourceFile

//PackageClause  = "package" PackageName .
//PackageName    = identifier .
packageClause
    : 'package' IDENTIFIER
    ;

importDecl
    : 'import' ( importSpec | '(' ( importSpec eos )* ')' )
    ;
```

Принцип работы генератора распознавателей языков программирования ANTLR 4



Пример грамматики

grammar Hello;

hello: 'hello' ID ;

ID : [a-z]+ ;

WS : [\t\r\n]+ -> skip ;

Использование сканера и распознавателя

```
public class HelloRunner {
    final static String fileName = "./Hello.txt";

    public static void main(String[ ] args) {
        try {
            // CharStream input = CharStreams.fromStream(System.in);
            CharStream input = CharStreams.fromFileName(fileName);
            HelloLexer lexer = new HelloLexer (input);

            CommonTokenStream tokens = new CommonTokenStream (lexer);

            HelloParser parser = new HelloParser ( tokens);
            ParseTree tree = parser.hello ();
            System.out.println(tree.toStringTree(parser));
        } catch (IOException e) { e.printStackTrace(); }
    }
}
```

Структура описания грамматики языка программирования

```
/** Комментарий для javadoc */  
{ lexer | parser }  
grammar имя_грамматики;  
options {...}  
import ... ;  
tokens {...}  
channels {...} // только для сканера  
@actionName {...}
```

Правила сканера и распознавателя

Настройка ANTLR (options)

```
grammar Name;
```

```
...
```

```
options {
```

```
    Language = Java;
```

```
    Output = AST;
```

```
    ASTLabelType = CommonTree;
```

```
}
```

```
...
```

Импорты ANTLR (options)

grammar Nested;

import G1, G2;

s : X ; ...

```
grammar G3;  
r : B ;
```

```
grammar G1;  
import G3;  
t : A ;
```

```
grammar G2;  
r : C ;
```

```
grammar Nested;  
import G1, G2;  
s : r ;
```



```
grammar Nested;  
s : r ; // from Nested  
r : B ; // from G3  
t : A ; // from G1
```

Лексемы (tokens). Пример

```
// Определение ключевых слов.  
tokens {  
    BEGIN, END, IF, THEN, WHILE  
}
```


Каналы (tokens). Пример

```
channels {  
    WSCHANNEL,  
    HIDDEN  
}
```

```
BLOCK_COMMENT : '/' .*? '/' -> channel(HIDDEN) ;  
LINE_COMMENT :  '/' ~[\r\n]* -> channel(HIDDEN) ;
```

```
// WS :      [ \t\r\n\f]+ -> skip ;  
WS :      [ \t\r\n\f]+ -> channel(HIDDEN);
```

Заголовок класса – распознавателя.

Пример

```
@header {  
package go.parser.generated;  
  
import java.util.*  
}
```

Члены класса – сканера (members).

Пример

```
@lexer::members {
```

```
// keywords map used in lexer to assign token types
```

```
Map<String,Integer> keywords =
```

```
    new HashMap<String,Integer>()
```

```
{
```

```
    put("begin", KeywordsParser.BEGIN);
```

```
    put("end", KeywordsParser.END);
```

```
    ...
```

```
};
```

```
}
```

Действия после прохода правила

```
grammar Count;
```

```
@header { package foo; }
```

```
@members { int count = 0; }
```

```
list
```

```
@after { System.out.println(count+" ints"); }
```

```
: INT { count++; }
```

```
( ',' INT { count++; }
```

```
)* ;
```

```
INT : [0-9]+ ;
```

```
WS : [ \r\t\n]+ -> skip ;
```

Действие *after* для правила list

```
grammar Count;
```

```
@header { package foo; }
```

```
@members { int count = 0; }
```

```
list
```

```
@after { System.out.println(count+" ints"); }
```

```
: INT { count++; }
```

```
( ',' INT { count++; }
```

```
)* ;
```

```
INT : [0-9]+ ;
```

```
WS : [ \r\t\n]+ -> skip ;
```

Правила сканера с альтернативами

TokenName

: альтернатива1

|

...

| альтернативаN

;

Правила сканера с альтернативами.

Пример

INT_LIT

: DECIMAL_LIT

| OCTAL_LIT

| HEX_LIT

;

Фрагменты правил сканера

FragmentName

fragment

: альтернатива1

|

...

| альтернативаN

;

Фрагменты правил сканера.

Пример

// Ссылается на вспомогательное правило **DIGIT**.

INT : **DIGIT**+ ;

fragment

DIGIT : [0-9] ; // Это не лексема

Элементы правил сканера.

Литералы

fragment

HexDigit : ('0'..'9'|'a'..'f'|'A'..'F')

Элементы правил сканера.

Множество СИМВОЛОВ

ID : [a-zA-Z] [a-zA-Z0-9]* ;

WS : [\n\r] -> skip ;

Элементы правил сканера.

Множество СИМВОЛОВ

COMMENT

:

'//' ~[\r\n]* '\r'? '\n' -> skip

;

Элементы правил сканера.

Диапазон символов

ID : LETTER (LETTER | '0'..'9')* ;

fragment

LETTER : [a-zA-Z\u0080-\u00FF_] ;

Элементы правил сканера.

Действие

```
END : ( 'endif' | 'end')
```

```
{ System.out.println("Нашли конец"); }
```

```
;
```

Элементы правил сканера.

Семантический предикат

stat: decl | expr ;

decl: ID ID ;

expr:

{ istype() }? ID '(' expr ')'

|

{ isfunc() }? ID '(' expr ')'

;

Правила распознавателя

Элементы правил распознавателя. Документирующий комментарий

```
/**
```

```
    Javadoc комментарий перед правилом
```

```
*/
```

```
returnStatement
```

```
    : 'return' expr ':'
```

```
    ;
```

Альтернативы правил распознавателя. Объявления верхнего уровня в *Go*

topLevelDecl

: declaration

| functionDecl

| methodDecl

;

Метки для альтернатив

expr

: expr '*' expr # BinaryOp

| expr '+' expr # BinaryOp

| INT # Int

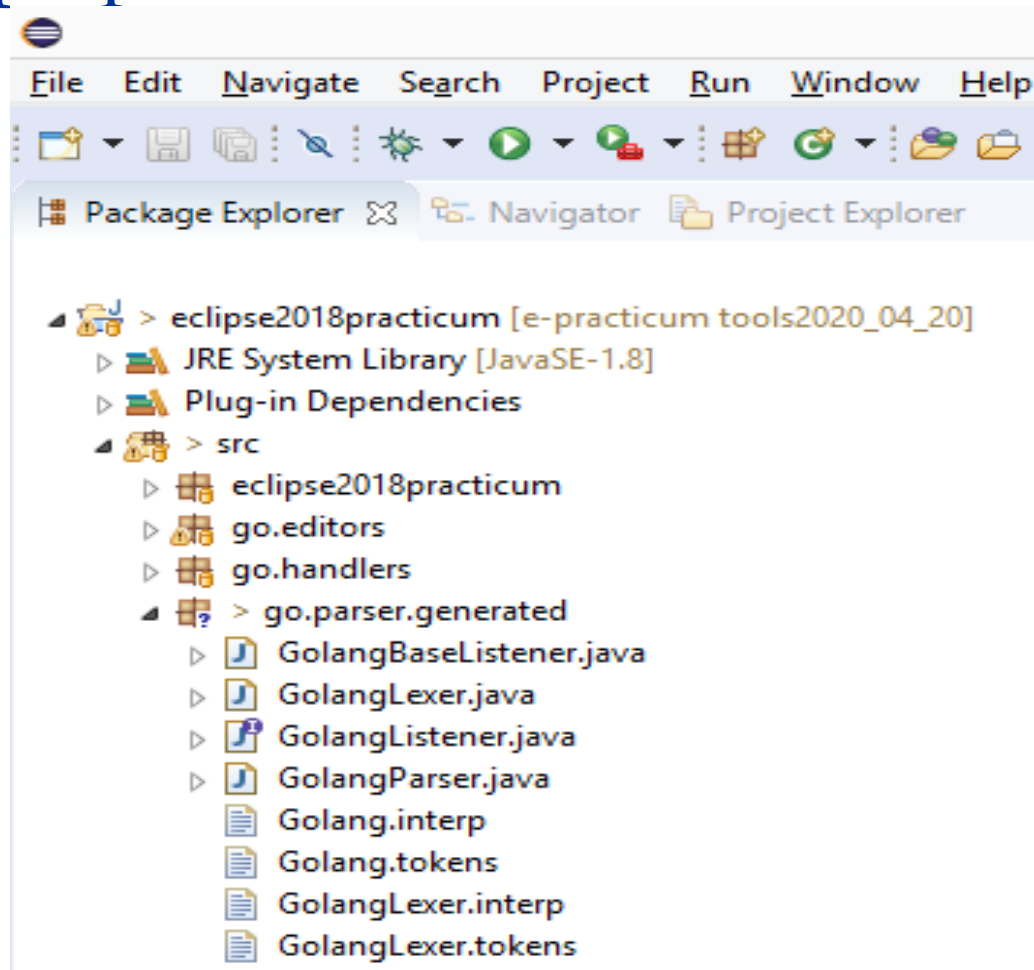
;

Метки для альтернатив

```
public interface AListener extends ParseTreeListener {  
  
    void enterBinaryOp(AParser.BinaryOpContext ctx);  
    void exitBinaryOp(AParser.BinaryOpContext ctx);  
  
    void enterInt(AParser.IntContext ctx);  
    void exitInt(AParser.IntContext ctx);  
}
```

Построение и анализ дерева программы на языке GoLang

Файлы распознавателя языка GoLang сгенерированные ANTLR



Файлы сгенерированные ANTLR.

Интерфейс *GoParserListener*

// Generated from Golang.g4 by ANTLR 4.8

```
package go.parser.generated;
```

```
import org.antlr.v4.runtime.tree.ParseTreeListener;
```

```
/**
```

```
* This interface defines a complete listener for a parse tree produced by
```

```
* {@link GoParser}.
```

```
*/
```

```
public interface GoParserListener extends ParseTreeListener
```

```
{
```

// Содержимое на следующем слайде ...

```
}
```

сгенерирован

Из файла
antlr-4.8-complete.jar

Файлы сгенерированные ANTLR.

Интерфейс *GoParserListener*

```
/**  
 * Enter a parse tree produced by {@link GoParser#sourceFile}.  
 * @param ctx the parse tree  
 */
```

```
void enterSourceFile(GolangParser.SourceFileContext ctx);
```

```
/**  
 * Exit a parse tree produced by {@link GoParser#sourceFile}.  
 * @param ctx the parse tree  
 */
```

```
void exitSourceFile(GoParser.SourceFile Context ctx);
```

```
170 sourceFile  
171     : packageClause eos ( importDecl eos )* ( topLevelDecl eos )*  
172     ;
```

```
/**  
 * Enter a parse tree produced by {@link GoParser#packageClause}.  
 * @param ctx the parse tree  
 */
```

```
void enterPackageClause(GoParser.PackageClauseContext ctx);
```

```
/**  
 * Exit a parse tree produced by {@link GoParser#packageClause}.  
 * @param ctx the parse tree  
 */
```

```
void exitPackageClause(GoParser.PackageClauseContext ctx);
```

Сгенерировано из
правила *sourceFile*

Сгенерировано из
правила *packageClause*

```
176 packageClause  
177     : 'package' IDENTIFIER  
178     ;
```


Файлы сгенерированные ANTLR.

Класс *GoParserBaseListener*

```
// Generated from Golang.g4 by ANTLR 4.8
```

```
package go.parser.generated;
```

```
import org.antlr.v4.runtime.ParserRuleContext;
```

```
import org.antlr.v4.runtime.tree.ErrorNode;
```

```
import org.antlr.v4.runtime.tree.TerminalNode;
```

```
/**  
 * This class provides an empty implementation of {@link GoParserListener},  
 * which can be extended to create a listener which only needs to handle a subset  
 * of the available methods.  
 */
```

```
public class GoParserBaseListener implements GoParserListener {
```

```
    // Содержимое на следующем слайде ...
```

```
}
```

сгенерирован

сгенерирован

Файлы сгенерированные ANTLR.

Класс *GoBaseListener*

```
/**  
 * {@inheritDoc}  
 *  
 * <p>The default implementation does nothing.</p>  
 */  
@Override public void enterSourceFile(GolangParser.SourceFileContext ctx) {}
```

Сгенерировано из
правила *sourceFile*

```
/**  
 * {@inheritDoc}  
 *  
 * <p>The default implementation does nothing.</p>  
 */  
@Override public void exitSourceFile(GolangParser.SourceFileContext ctx) {}
```

Сгенерировано из
правила *packageClause*

```
/**  
 * {@inheritDoc}  
 *  
 * <p>The default implementation does nothing.</p>  
 */  
@Override public void enterPackageClause(GolangParser.PackageClauseContext ctx) {}
```

```
/**  
 * {@inheritDoc}  
 *  
 * <p>The default implementation does nothing.</p>  
 */  
@Override public void exitPackageClause(GolangParser.PackageClauseContext ctx) {}
```

Распознаватель языка GoLang (1)

```
public class GolangRunner {  
    static  
    public void parseFile (IFile file, IMessageHandler mh,  
                           IUMLBuilder umlBuilder)  
    {  
        String fileName = file.getLocation().toString();  
        try {  
            CharStream input = CharStreams.fromFileName(fileName);  
            GoLexer lexer = new GoLexer(input);  
            CommonTokenStream tokens = new CommonTokenStream(lexer);  
  
            //...
```

сгенерирован

Распознаватель языка GoLang (2)

//...

сгенерирован

```
GoParser parser = new GoParser(tokens);
```

```
ANTLRErrorListener errorListener = new GolangErrorListener(mh);  
parser.addErrorListener(errorListener);
```

```
ParseTree tree = parser.sourceFile();
```

Начало распознавания с
первого правила языка
Golang 'sourceFile'.

```
ParseTreeWalker walker = new ParseTreeWalker();
```

сгенерирован

```
GoParserBaseListener goListener =  
    new GolangTreeListener(parser, umlBuilder);
```

```
walker.walk(goListener, tree);
```

```
}
```

Выдача диагностики в Eclipse (5)

```
public class GolangErrorListener extends BaseErrorListener {
    private IMessageHandler mh;

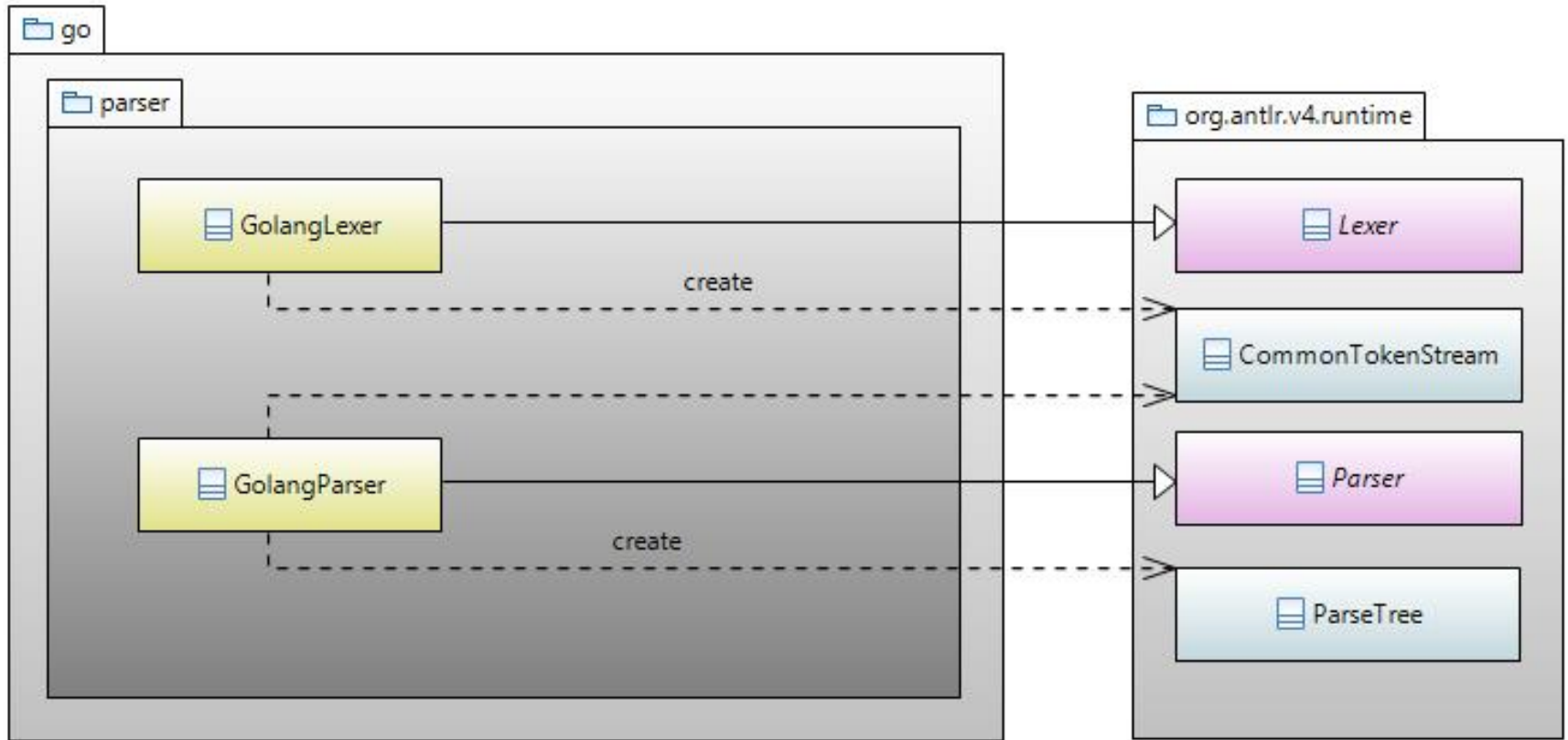
    public GolangErrorListener(IMessageHandler mh) { this.mh = mh; }

    @Override
    public void syntaxError (Recognizer<?, ?> recognizer,
        Object offendingSymbol,
        int line, int column, String message,
        RecognitionException e)
    {
        String fileName = recognizer.getInputStream().getSourceName();

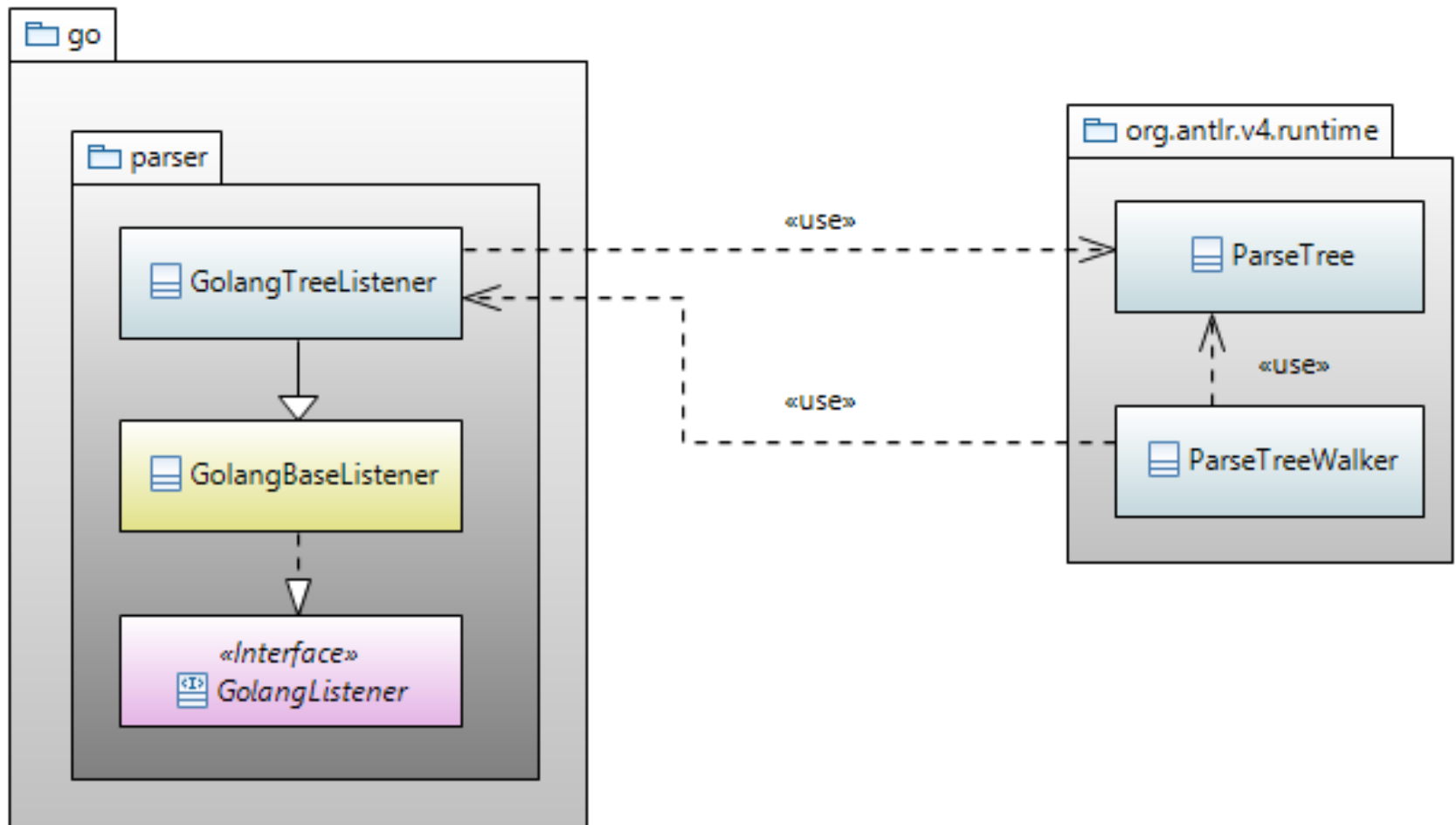
        ParseMessage m = new ParseMessage();
        m.fileName = fileName;
        m.line = line;
        m.column = column;
        m.message = message;

        mh.error(m);
    }
}
```

Классы распознавателя языка Go (1)



Классы распознавателя языка Go (2)



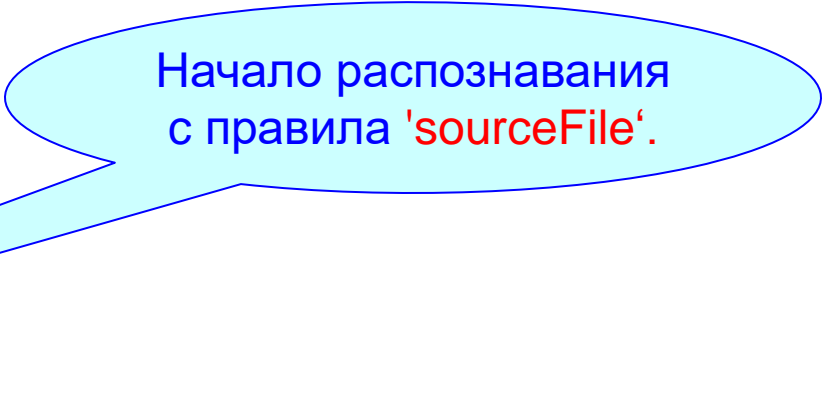
Правила языка **Go**. *SourceFile*

```
grammar GoParser;
```

```
@parser::members {  
// ...  
}
```

```
@lexer::members {  
//...  
}
```

```
// SourceFile = PackageClause ";" { ImportDecl ";" } { TopLevelDecl ";" } .  
sourceFile  
: packageClause eos ( importDecl eos )* ( topLevelDecl eos)*  
;
```



Начало распознавания
с правила 'sourceFile'.

Правила языка **Go**. *PackageClause*

// PackageClause = "package" PackageName .

// PackageName = identifier .

packageClause

: 'package' *IDENTIFIER*

;

Контекст правила *PackageClause*

```
/**
packageClause
    : 'package' IDENTIFIER
    ;
*/
public static class PackageClauseContext extends ParserRuleContext {
    public TerminalNode IDENTIFIER()

    public PackageClauseContext(ParserRuleContext parent, int invokingState);

    @Override public int getRuleIndex();
    @Override public void enterRule(ParseTreeListener listener);
    @Override public void exitRule(ParseTreeListener listener);
}
```

Правила языка **Go**. *PackageClouse*

```
public class GolangTreeListener extends GolangBaseListener {  
  
    private IUMLBuilder umlBuilder;  
  
    public GolangTreeListener(GolangParser parser, IUMLBuilder umlBuilder) {  
        this.umlBuilder = umlBuilder;  
    }  
}
```

Правила языка Go. *PackageClause*

```
public class GolangTreeListener extends GolangBaseListener {
private IUMLBuilder umlBuilder;
/**
packageClause
    : 'package' IDENTIFIER
    ;
*/
@Override
public void enterPackageClause (GolangParser.PackageClauseContext ctx) {
    final String name = ctx.IDENTIFIER().getText();

    umlBuilder.addPackage(name);

    System.out.format("package %s { %n", ctx.IDENTIFIER());
}
}
```

Правила языка **Go**. *TypeSpec*

typeDecl

: *TYPE* (*typeSpec* | *L_PAREN* (*typeSpec* eos)* *R_PAREN*)

;

typeSpec

: aliasDecl

| typeDef

;

aliasDecl

: *IDENTIFIER* *ASSIGN* *type_*

;

typeDef

: *IDENTIFIER* *typeParameters?* *type_*

;

Контекст правила *TypeSpec*

```
/**
typeDef
    : IDENTIFIER typeParameters? type_ ;
type_ : typeName typeArgs? | typeLit | L_PAREN type_ R_PAREN ;
typeArgs : L_BRACKET typeList COMMA? R_BRACKET ;
typeName : qualifiedIdent | IDENTIFIER ;
typeLit : arrayType | structType | pointerType | functionType | interfaceType
        sliceType | mapType | channelType ;
*/
public static class TypeDefContext extends ParserRuleContext {
    public TerminalNode IDENTIFIER()
        { return getToken(GoParser.IDENTIFIER, 0); }
    public Type_Context type_()
        { return getRuleContext(Type_Context.class,0); }
    public TypeParametersContext typeParameters()
        { return getRuleContext(TypeParametersContext.class,0); }
    public TypeDefContext(ParserRuleContext parent, int invokingState)
        { super(parent, invokingState); }
}
```

Контекст правила *TypeDef*

```
/**
    typeSpec : aliasDecl | typeDef ;
    aliasDecl : IDENTIFIER ASSIGN type_ ;
    typeDef : IDENTIFIER typeParameters? type_ ;
    type_ : typeName typeArgs? | typeLit | L_PAREN type_ R_PAREN ; typeArgs :
    L_BRACKET typeList COMMA? R_BRACKET ;
    typeName : qualifiedIdent | IDENTIFIER ;
    typeLit : arrayType | structType | pointerType | functionType | interfaceType
    | sliceType | mapType | channelType ;
*/
@Override
public void enterTypeDef(GoParser.TypeDefContext ctx) {
    final String name = ctx.IDENTIFIER().getText();

    Type_Context t = ctx.type_();
    System.out.format("type %s { %n", name);

    // ...
}
```

Контекст правила *TypeDef*

// ...

```
TypeLitContext typeLit = t.typeLit();
```

```
if (typeLit == null)
```

```
    return;
```

```
InterfaceTypeContext i_literal = typeLit.interfaceType();
```

```
if (i_literal != null) {
```

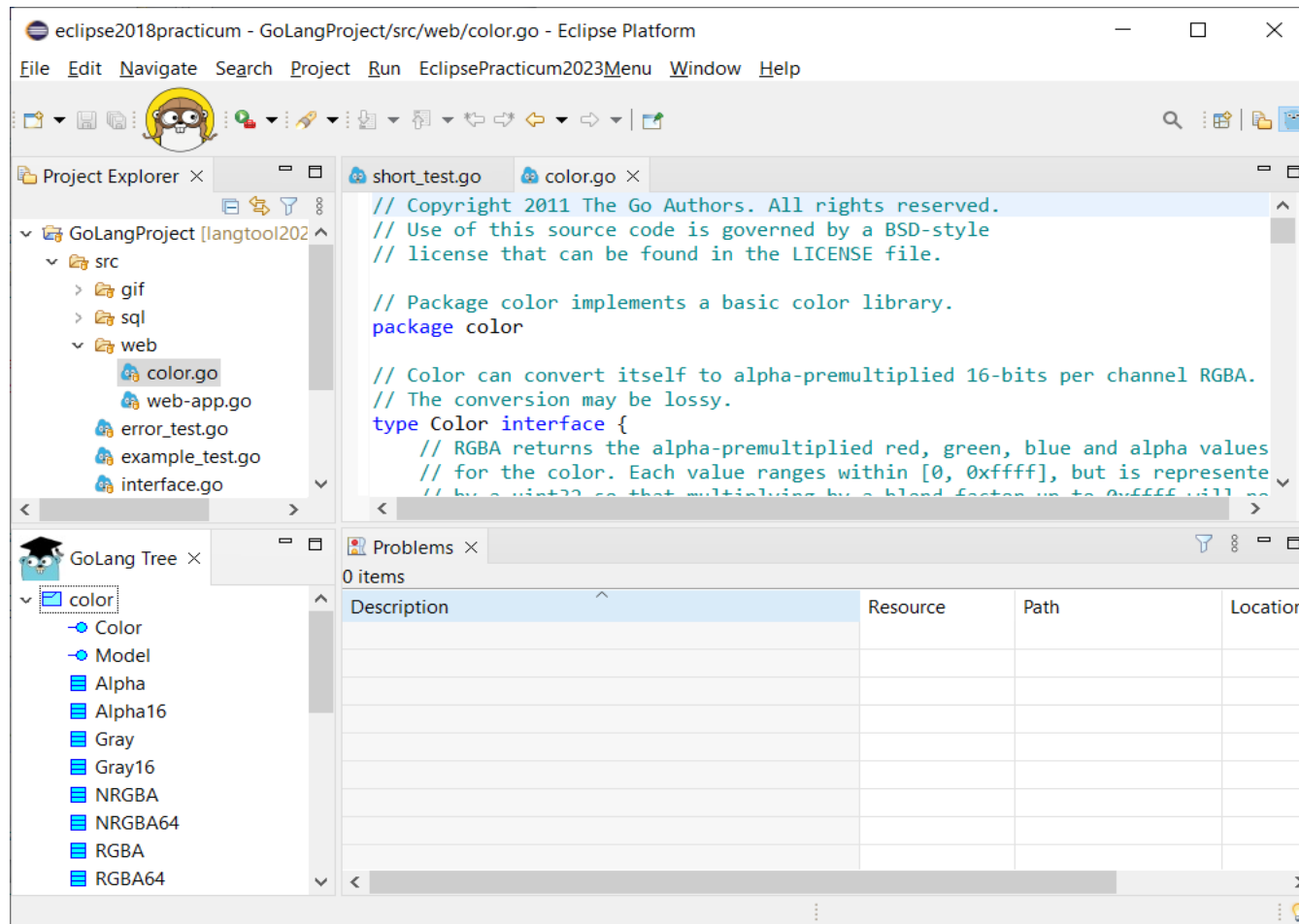
```
    umlBuilder.addInterface(name);
```

```
    return;
```

```
}
```

```
}
```


Пакеты и классы языка Go в UML-модели программы



Построение диаграммы зависимости пакета

Диаграмма зависимости пакета Badger

The screenshot displays the Eclipse IDE interface for a Go project named 'badger'. The main editor shows the 'main.go' file with the following code:

```
1 package badger
2
3 import (
4     "bufio"
5     "encoding/binary"
6     "io"
7     "log"
8     "sync"
9
10    "github.com/dgraph-io/badger/y"
```

The UML View window shows a dependency diagram for the 'badger' package. The diagram consists of the following elements:

- A package named 'cmd' has a dashed dependency arrow pointing to the 'badger' package.
- The 'badger' package has a dashed dependency arrow pointing to the 'bufio' package.
- The 'badger' package has a dashed dependency arrow pointing to the 'binary' package.
- The 'badger' package has a dashed dependency arrow pointing to the 'io' package.
- The 'badger' package has a dashed dependency arrow pointing to the 'log' package.
- The 'badger' package has a dashed dependency arrow pointing to the 'sync' package.

The Project Explorer on the left shows the project structure, including sub-packages like 'cmd', 'contrib', 'images', 'options', 'protos', 'skl', 'table', and 'y', as well as files like 'appveyor.yml', 'backup_test.go', 'backup_test.go.txt', and 'backup.go'. The Package Explorer at the bottom left shows the internal structure of the 'badger' package, including 'DB', 'Entry', 'Item', 'Iterator', 'IteratorOptions', and 'MessageDB'.