

컴파일러개론 7주차 실습

PRETTY PRINT USING LISTENER

2024. 10. 25.

TA: 박정필

✉: 202350941@o.cnu.ac.kr

공지, 질문 방법

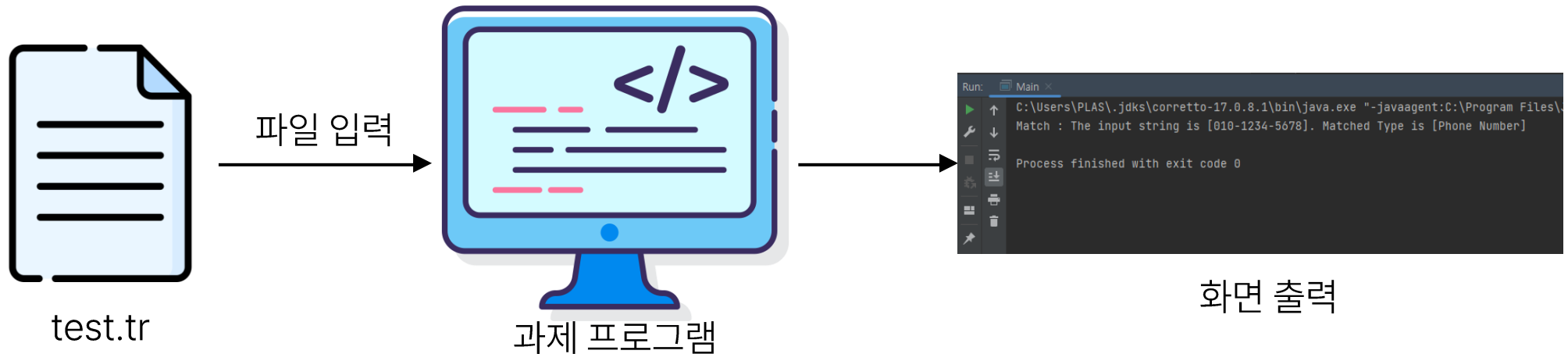
- 강의자료, 동영상
 - 사이버캠퍼스 업로드 예정
- 공지
 - 카카오톡 오픈채팅, 사이버캠퍼스
 - <https://open.kakao.com/o/gkfF7JMg>
 - 오픈프로필 사용 가능
 - 참여코드: 2024cp01
- 질문
 - 수업시간, 카카오톡 오픈채팅에 질문
 - 과제 관련 질문은 제출기한 전날까지만 가능 (당일 질문은 답변 X)
 - 이메일
 - 이메일 제목은 "[컴파일러개론][분반] ...", 제목 반드시 준수
 - 교수님 : eschough@cnu.ac.kr
 - TA : 202350941@o.cnu.ac.kr

목차

- 7주차 과제
 - tinyR4 Pretty Print using Listener

7주차 과제: tinyR4 Pretty Print

- Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드를 listener를 활용하여 작성하기
 - 구조



7주차 과제: tinyR4 Pretty Print

- Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드 작성하기
 - 결과

```
fn main() {  
    let a = 3;  
    let b = 4;  
        c = a + b;  
    d = 3 - (4 + 5);  
    println!(c);  
    println!("Hello World");  
}  
  
fn sum(a:u32, b:u32) -> u32 {  
    return a + b;  
}
```

test.tr

```
fn main () {  
    let  a  = 3;  
    let  b  = 4;  
    c = a + b;  
    d = 3 - (4 + 5);  
    println!(c);  
    println!("Hello World");  
}  
fn sum (a:u32, b:u32) -> u32{  
    return a + b;  
}
```

결과 화면 출력

7주차 과제: tinyR4 Pretty Print

- Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드 작성하기 (조건)
 1. 블록이나 nesting 되어 들어갈 때는 4칸 들여쓰기
 2. 2진 연산자와 피 연산자 사이에는 빈칸 1칸
 3. Assignment는 let 키워드를 사용하여 작성
 4. Type은 u32로 고정
 5. 사이버 캠퍼스에 올라온 tinyR4.g4 문법 파일을 사용
 6. 그 외의 프로젝트 설정은 2주차 실습 자료 참고
 7. Listener 방식을 이용하여 코드 구성

7주차 과제: tinyR4 Pretty Print

■ Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드 작성하기

1. 블록이나 nesting 되어 들어갈 때는 4칸 들여쓰기 (아래는 예시로 ^로 표시)
 - 중첩 될 수 있음
 - fn, if, else, assignment, ... 등등

```
fn SUM(x:u32, y:u32) -> u32 {  
^^^^return x + y;  
}
```

```
fn FOO(x:u32, y:u32) -> u32 {  
^^^^If x == y {  
^^^^^^^^println!("hello world");  
^^^^} else {  
^^^^^^^^println!("test");  
^^^^}  
^^^^return 1;  
}
```

7주차 과제: tinyR4 Pretty Print

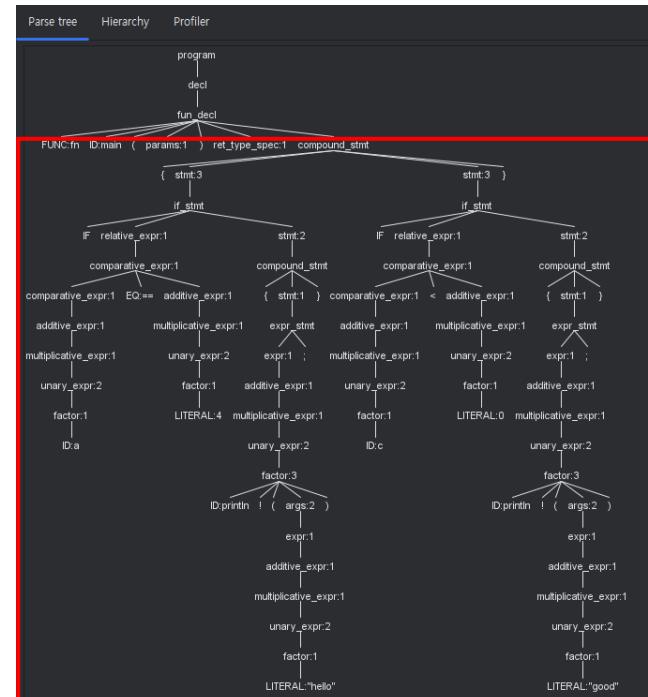
- Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드 작성하기
 1. 블록이나 nesting 되어 들어갈 때는 4칸 들여쓰기 (아래는 예시로 ^로 표시)
 - 중첩 계산시 Newline 으로 구분되는 것을 parse tree로 확인하기

tinyR4.g4 start rule: program

Input ☐ File

```

1 fn main() {
2     if a == 4 {
3         println!("hello");
4     }
5     if c < 0 {
6         println!("good");
7     }
8 }
  
```



7주차 과제: tinyR4 Pretty Print

- Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드 작성하기
 - 2. 2진 연산자와 피 연산자 사이에는 빈칸 1칸
 - $X + Y$;
 - 3. Assignment는 let 키워드를 추가하여 작성. 단, 계산 후 Assignment는 금지
 - `let A = 3; <O>`
 - `let a = 3 + x; <X>`
 - 2진 연산자(or stmt) 후에 Assignment는 불가
(g4 순서상 local_decl 후에 stmt가 오기 때문)
∴ Assignment 후에 2진 연산자(or stmt)를 작성
 - 3. Type은 u32로 고정

7주차 과제: tinyR4 Pretty Print

- Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드 작성하기
 - 4. 사이버 캠퍼스에 올라온 tinyR4.g4 문법 파일을 사용
 - 문법 제약이 있을 수 있음을 확인 -> 메일 및 오픈톡방 질문 활용

```

27 val      : literal
28     | id
29     ;
30
31 stmt      : expr_stmt
32     | compound_stmt
33     | if_stmt
34     | for_stmt
35     | return_stmt
36     | break_stmt
37     | loop_stmt
38     ;
39
40 expr_stmt : expr ';' ;
41
42 expr      : additive_expr
43     | relative_expr
44     | id '=' expr
45     ;

```

```

72 if_stmt   : IF relative_expr compound_stmt (ELSE compound_stmt)? ;
73
74 for_stmt  : FOR id 'in' range compound_stmt ;
75
76 loop_stmt : LOOP compound_stmt;
77
78 range     : literal '..' ('=')? literal ;
79
80 return_stmt : RETURN (expr)? ';' ;
81
82 break_stmt  : BREAK ';' ;
83
84 args       :
85     | expr (',' expr)* ;
86
87 literal    : LITERAL;
88
89 id         : ID;

```

7주차 과제: tinyR4 Pretty Print

- Java, ANTLR 를 이용하여 tinyR4 문법으로 된 코드에 대한 pretty print 코드 작성하기
 - 과제 입력 파일 조건
 - Parsing 이 잘되는 코드
 - 에러가 없는 코드
 - 라인 별로 구분된 코드

```
fn main() {  
    let a = 3;  
    let b = 4;  
        c = a + b;  
d = 3 - (4 + 5);  
println!(c);  
println!("Hello World");  
}  
  
fn sum(a:u32, b:u32) -> u32 {  
    return a + b;  
}
```

7주차 과제: tinyR4 Pretty Print

■ Background

- 코드를 이쁘게 출력하기 위해 생성된 parse tree를 순회하면서 값을 출력
- 순회 방법은 Listener, Visitor 두 가지가 존재하나 그 중 Listener 만 생각
- 2주차 실습 내용의 문법과 흡사하지만, 방법이 다름.

```

1 grammar tinyR4;
2
3 program      : decl+      {System.out.println("202200000 Rule 0");} ;
4
5 decl         : fun_decl {System.out.println("202200000 Rule 1");};
6
7 fun_decl     : FUNC ID '(' params ')' ret_type_spec compound_stmt {System.out.println("202200000 Rule 2");} ;
8
9 params       : {System.out.println("202200000 Rule 3-1");}
10 | param (',' param)* {System.out.println("202200000 Rule 3-2");}
11 ;
  
```

2주차 실습 문법(rule 번호 O)

```

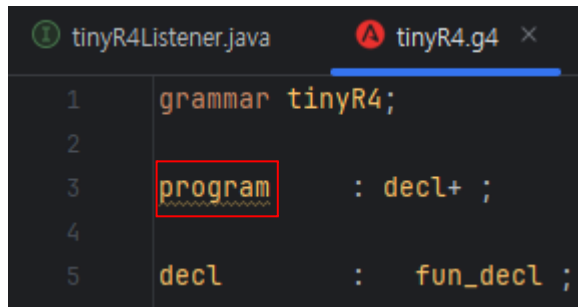
program      : decl+ ;
decl         : fun_decl ;
fun_decl     : FUNC ID '(' params ')' ret_type_spec compound_stmt ;
params       :
| param (',' param)*
;
  
```

2주차 실습 문법 (rule 번호 X)

7주차 과제: tinyR4 Pretty Print

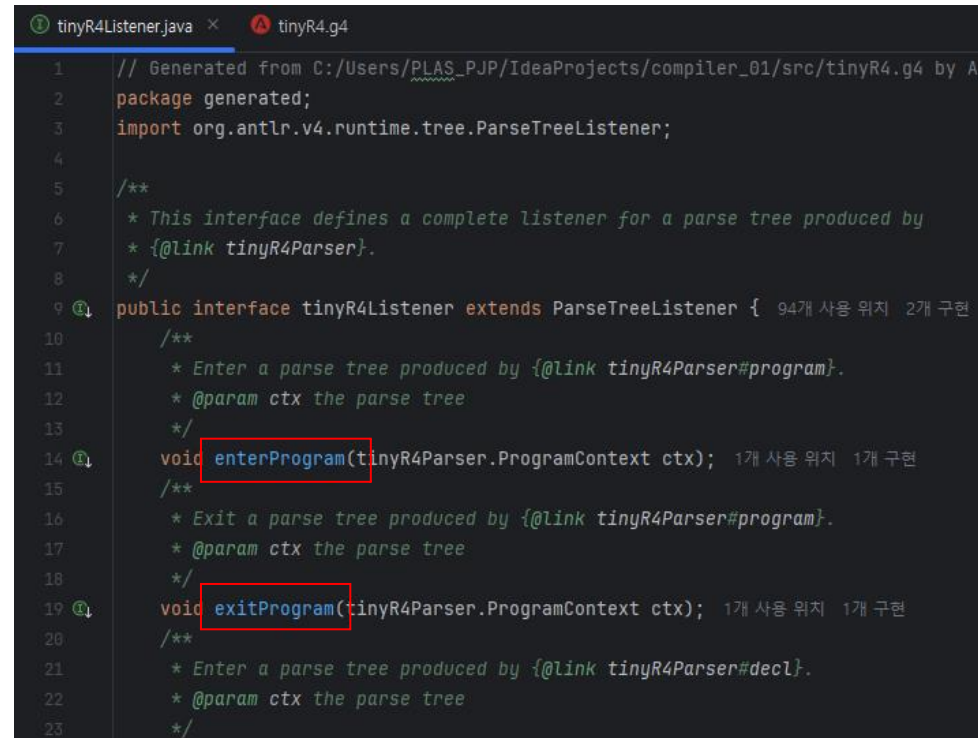
■ Background

- Listener 방식
- Non-Terminal node를 만날 경우 enter, exit 두 함수를 실행하여 노드 방문을 확인



```

1 grammar tinyR4;
2
3 program      : decl+ ;
4
5 decl        : fun_decl ;
  
```



```

1 // Generated from C:/Users/PLAS_PJP/IdeaProjects/compiler_01/src/tinyR4.g4 by ANTLR
2 package generated;
3 import org.antlr.v4.runtime.tree.ParseTreeListener;
4
5 /**
6  * This interface defines a complete listener for a parse tree produced by
7  * {@link tinyR4Parser}.
8  */
9 public interface tinyR4Listener extends ParseTreeListener {
10     /**
11      * Enter a parse tree produced by {@link tinyR4Parser#program}.
12      * @param ctx the parse tree
13      */
14     void enterProgram(tinyR4Parser.ProgramContext ctx);
15     /**
16      * Exit a parse tree produced by {@link tinyR4Parser#program}.
17      * @param ctx the parse tree
18      */
19     void exitProgram(tinyR4Parser.ProgramContext ctx);
20     /**
21      * Enter a parse tree produced by {@link tinyR4Parser#decl}.
22      * @param ctx the parse tree
23      */
  
```

7주차 과제: tinyR4 Pretty Print

■ ParserRuleContext (tinyR4PrintListener.java)

- ctx를 통해 .g4 파일의 문법에 접근
- ParseTreeProperty 객체를 생성하여 각 문법에 접근
- ParseTreeProperty r4Tree 객체를 통한 예시 (객체 이름은 자유)
 - `r4Tree.get(ctx.fun_decl())` = ParseTree에서 `ctx.fun_decl()` 문법에 해당하는 문자열을 가져옴
 - `r4Tree.put(ctx, fun_decl)` = ParseTree에 `ctx.fun_decl()` 문법에 해당하는 문자열 `fun_decl`를 저장함
- 주로 `exit`가 붙어있는 함수를 Override하여 구현하며, 각 `exit`가 붙어있는 함수는
 - 먼저 하는 일 → 자식 노드 String = `r4Tree.get(자식 노드)` 확보
 - 끝에 하는 일 → `r4Tree.put(자기 자신 노드, 자기 자신 String)`으로 저장

ParseTreeProperty r4Tree 객체 →

r4Tree 객체를 이용한
`exitDecl` 함수 구현 예시→

```
ParseTreeProperty<String> r4Tree = new ParseTreeProperty<>();

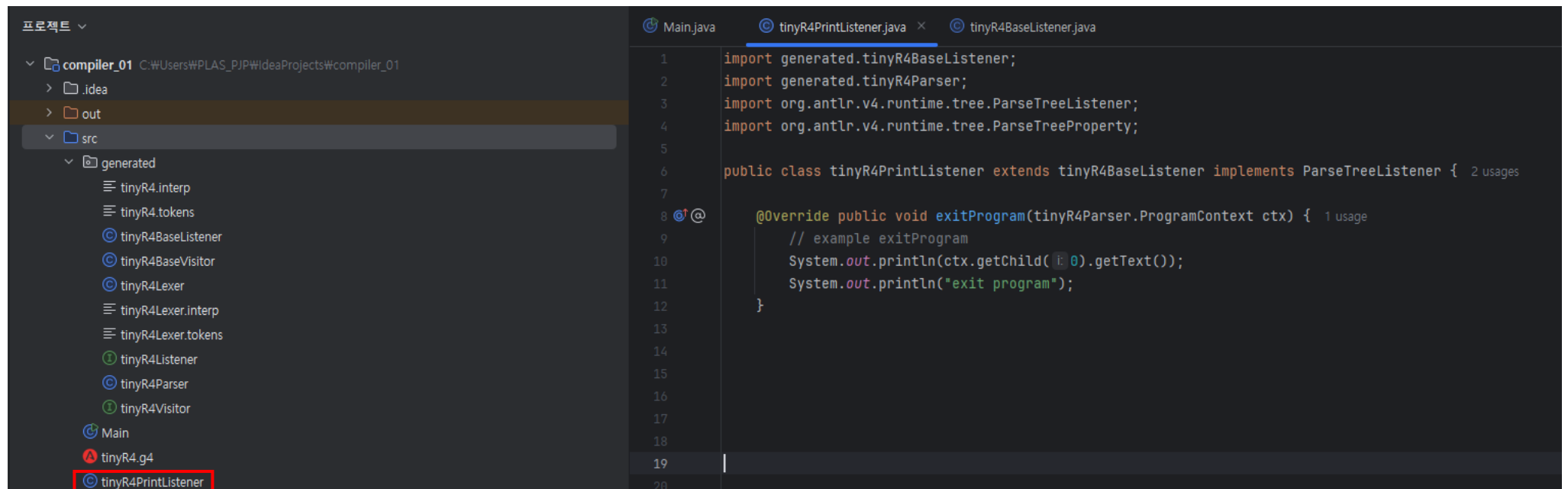
@Override public void exitDecl(tinyR4Parser.DeclContext ctx) {
    String fun_decl = r4Tree.get(ctx.fun_decl());
    r4Tree.put(ctx, fun_decl);
}
```

7주차 과제: tinyR4 Pretty Print

- 프로젝트 설정 추가
 - prettyPrint.java 작성

7주차 과제: tinyR4 Pretty Print

- prettyPrint.java (tinyR4PrintListener.java) 생성
 - tinyR4BaseListener를 상속 받는 하위 클래스로 생성
 - tinyR4BaseListener는 tinyR4Listener 인터페이스(스켈레톤) 코드를 상속 받음
 - 위의 두 코드를 참고 하여 각 함수들을 상속받아서 구현



The screenshot shows an IDE with a project named 'compiler_01'. The project structure on the left includes a 'src' directory with a 'generated' subdirectory. The 'generated' directory contains several files, including 'tinyR4PrintListener.java', which is highlighted with a red box. The main editor displays the code for 'tinyR4PrintListener.java', which imports 'generated.tinyR4BaseListener', 'generated.tinyR4Parser', 'org.antlr.v4.runtime.tree.ParseTreeListener', and 'org.antlr.v4.runtime.tree.ParseTreeProperty'. The code defines a public class 'tinyR4PrintListener' that extends 'tinyR4BaseListener' and implements 'ParseTreeListener'. It includes an '@Override' method 'exitProgram' that prints the text of the current node and its children.

```

1  import generated.tinyR4BaseListener;
2  import generated.tinyR4Parser;
3  import org.antlr.v4.runtime.tree.ParseTreeListener;
4  import org.antlr.v4.runtime.tree.ParseTreeProperty;
5
6  public class tinyR4PrintListener extends tinyR4BaseListener implements ParseTreeListener { 2 usages
7
8  @Override public void exitProgram(tinyR4Parser.ProgramContext ctx) { 1 usage
9      // example exitProgram
10     System.out.println(ctx.getChild(0).getText());
11     System.out.println("exit program");
12 }
13
14
15
16
17
18
19
20

```


7주차 과제: tinyR4 Pretty Print

- prettyPrint.java (tinyR4PrintListener.java) 생성
 - 상속받은 ctx 자료구조에 대해서는 ANTLR API 문서 참고
 - [ParserRuleContext \(ANTLR 4 Runtime 4.13.1 API\)](#)

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

ALL CLASSES SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package org.antlr.v4.runtime

Class ParserRuleContext

java.lang.Object
 org.antlr.v4.runtime.RuleContext
 org.antlr.v4.runtime.ParserRuleContext

All Implemented Interfaces:
 ParseTree, RuleNode, SyntaxTree, Tree

Direct Known Subclasses:
 InterpreterRuleContext, RuleContextWithAltNum

public class **ParserRuleContext**
 extends RuleContext

A rule invocation record for parsing. Contains all of the information about the current rule not stored in the RuleContext. It handles parse tree children list, Any ATN state tracing, and the default values available for rule invocations: start, stop, rule index, current alt number. Subclasses made for each rule and grammar track the parameters, return values, locals, and labels specific to that rule. These are the objects that are returned from rules. Note text is not an actual field of a rule return value; it is computed from start and stop using the input stream's toString() method. I could add a ctor to this so that we can pass in and store the input stream, but I'm not sure we want to do that. It would seem to be undefined to get the .text property anyway if the rule matches tokens from multiple input streams. I do not use getters for fields of objects that are used simply to group values such as this aggregate. The getters/setters are there to satisfy the superclass interface.

7주차 과제: tinyR4 Pretty Print

■ main 코드 작성

- Walker: 생성된 parse tree를 depth-first로 순회하면서 Listener 동작 수행

```
Main.java x
1  import org.antlr.v4.runtime.*;
2  import org.antlr.v4.runtime.tree.ParseTreeWalker;
3  import org.antlr.v4.runtime.tree.ParseTree;
4  import generated.*;
5
6  public class Main {
7      public static void main(String[] args) throws Exception {
8          CharStream code = CharStreams.fromFileName("./src/rust.tr");
9          tinyR4Lexer lexer = new tinyR4Lexer(code);
10         CommonTokenStream tokens = new CommonTokenStream(lexer);
11
12         try {
13             tinyR4Parser parser = new tinyR4Parser(tokens);
14             ParseTree tree = parser.program();
15
16             ParseTreeWalker walker = new ParseTreeWalker();
17             walker.walk(new tinyR4PrintListener(), tree);
18
19             System.out.println(tinyR4PrintListener.getOutput());
20         } catch (RuntimeException e) {
21             System.out.println("Error");
22         }
23     }
24 }
```

7주차 과제: tinyR4 Pretty Print

■ main 코드 작성

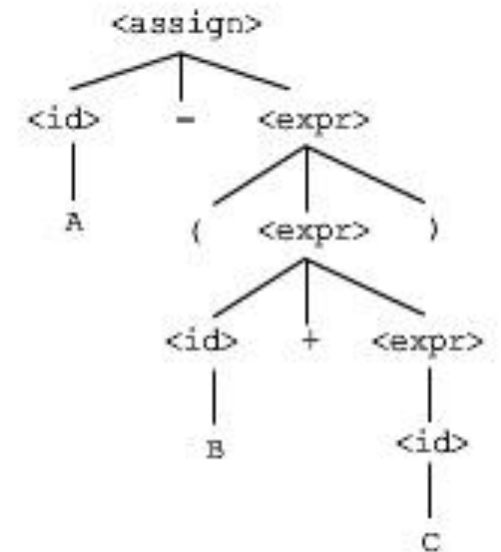
- Walker: 생성된 parse tree를 depth-first로 순회하면서 Listener 동작 수행

• Depth first 방문

- `<assign>` 노드 방문 → `<id>` 노드 방문 → A → - → `<expr>` 노드 방문 → (→ `<expr>` 노드 방문 → `<id>` 노드 방문 → B → + → `<expr>` 노드 방문 → `<id>` 노드 방문 → C →)

• Listener 호출 순서

`<assign>` enter → `<id>` enter → A → `<id>` exit → - → `<expr>` enter → (→ `<expr>` enter → `<id>` enter → B → `<id>` exit → + → `<expr>` enter → `<id>` enter → C → `<id>` exit → `<expr>` exit →) → `<expr>` exit → `<assign>` exit



Parse Tree for the Statement A = (B + C)

7주차 과제: tinyR4 Pretty Print

■ 과제 세부사항

- 이전 실습 과제들 동일한 방식으로,
 - 사이버캠퍼스 7주차 과제 란에 제출
 - 여러 개 파일로 구현 가능, 하지만 package 사용 금지
 - main 함수 있는 파일 이름은 **Main.java**
 - .java 파일들 **학번.zip**으로 압축해서 제출
- 코드에 주석 잘 달기 (필수 X)
- 입력 파일은 **test.tr**
- Visitor로 구현하면 0점
- 과제 제출 프로젝트의 트리 구조는 다음 페이지 참고 (src 폴더 압축)

■ 마감 (기간 준수)

- **2024년 11월 1일 금요일 23시 59분** (기한 엄수)
- 추가 제출 기한 없음

7주차 과제: tinyR4 Pretty Print

■ 과제 구현 참고

- 이번 7주차 실습은 5주차 실습인 Ugly Print의 연장선으로, 나머지 문법에 대해 작성해야된다.
- 10월 26일 (토)에 완성된 Ugly Print 코드를 배포할 예정입니다.
해당 코드를 참고하여 작성하시면 됩니다.

7주차 과제: tinyR4 Pretty Print

■ 과제 구현 참고 예제 1

```

1  fn main() {
2      let a = 10;
3      foo(a);
4      return 0;
5  }
6
7  fn foo(num:u32) -> u32 {
8      let mut result = 0;
9      for i in 0..10 {
10         if(i > (num / 2)) {
11             println!(i);
12             result = i;
13             break;
14         }
15     }
16     return i;
17 }

```



```

fn main() {
    let a = 10;
    foo(a);
    return 0;
}

fn foo(num:u32) -> u32 {
    let mut result = 0;
    for i in 0..10 {
        if (i > (num / 2)) {
            println!(i);
            result = i;
            break;
        }
    }
    return i;
}

```

7주차 과제: tinyR4 Pretty Print

■ 과제 구현 참고 예제 2

```

1  fn main() {
2      let c = a;
3      a + b;
4
5      if (a < 10) {
6          println!("a");
7      } else {
8          println!(123);
9      }
10     for i in 1..4 {
11         println!(i);
12     }
13     return 0;
14 }

```



```

fn main() {
    let c = a;
    a + b;
    if (a < 10) {
        println!("a");
    } else {
        println!(123);
    }
    for i in 1..4 {
        println!(i);
    }
    return 0;
}

```

7주차 과제: tinyR4 Pretty Print

- 5주차 Ugly Print의 exitExpr, exitAdditive_expr 함수 구현 예시

```
@Override public void exitExpr(tinyR4Parser.ExprContext ctx) { 1개 사용 위치
    String result = "";
    if(ctx.additive_expr() != null)
        result = r4Tree.get(ctx.additive_expr());
    else if(ctx.relative_expr() != null)
        result = r4Tree.get(ctx.relative_expr());
    r4Tree.put(ctx, result);
}

@Override public void exitAdditive_expr(tinyR4Parser.Additive_exprContext ctx) {
    String result = "";
    if(ctx.additive_expr() != null) {
        String left = r4Tree.get(ctx.additive_expr());
        String op = ctx.getChild(1).getText();
        String right = r4Tree.get(ctx.multiplicative_expr());
        result = left + " " + op + " " + right;
    } else
        result = r4Tree.get(ctx.multiplicative_expr());
    r4Tree.put(ctx, result);
}
```


7주차 과제: tinyR4 Pretty Print

■ 과제 디렉토리 구조

- tinyR4.g4 generated한 파일
 - 4개의 필수 파일
 - tinyR4BaseListener, tinyR4Lexer, tinyR4Listener, tinyR4Parser
- Main.java
- tinyR4.g4
- tinyR4PrintListener (Ugly에서 업그레이드)

