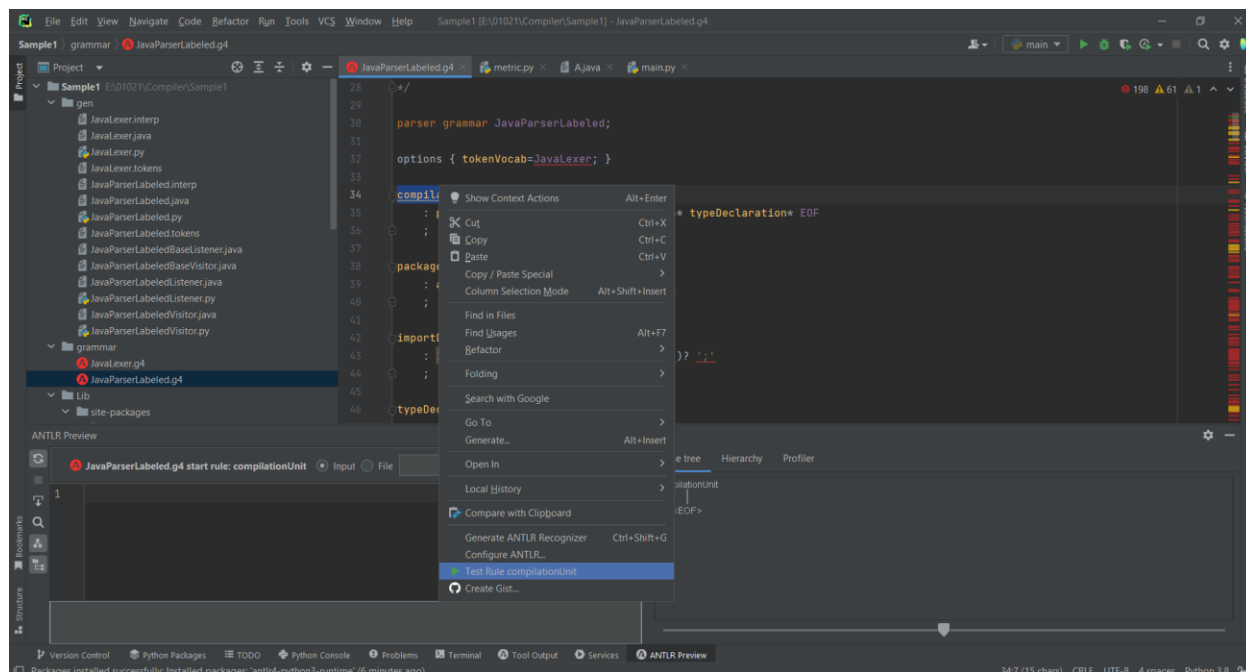


## سوال پنجم:

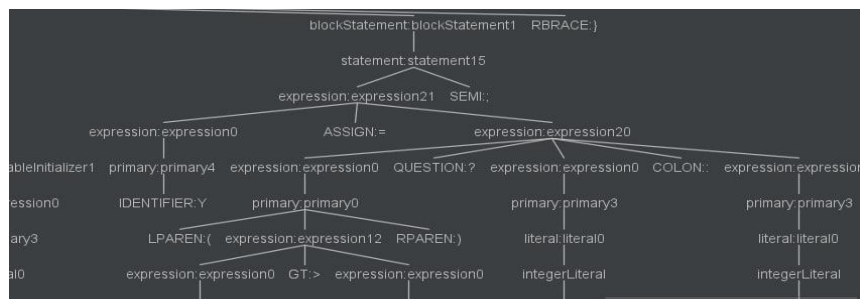
فایل های گرامر جاوا را که از قبل داشتیم در فولدر grammar قرار دادیم، سپس با کلیک راست روی فایل های گرامر و کلیک روی Generate ANTLR Recognition فایل های lexer, parser, listener, visitor را در فولدر gen تولید کردیم.

برای تشکیل درخت تجزیه ابتدا فایل JavaParserLabeled را باز کردیم سپس روی تابع compilationUnit کلیک راست کردیم و گزینه ی مشخص شده در تصویر زیر را انتخاب کردیم. پایین صفحه یک پنجره باز میشود که در سمت چپ آن میتوانیم کد جاوا را بنویسیم و در سمت راست درخت تجزیه ی گرامر آن را مشاهده کنیم.



کد زیر را در قسمت پایین و سمت چپ وارد کردیم تا ببینیم کدا توابع باید برای inline statement گفته شده Override شوند:

```
public class Main
{
    public solve(){
        int h = 0;
        int Y = 0;
        Y = (h>0) ? 1 : 3;}
}
```



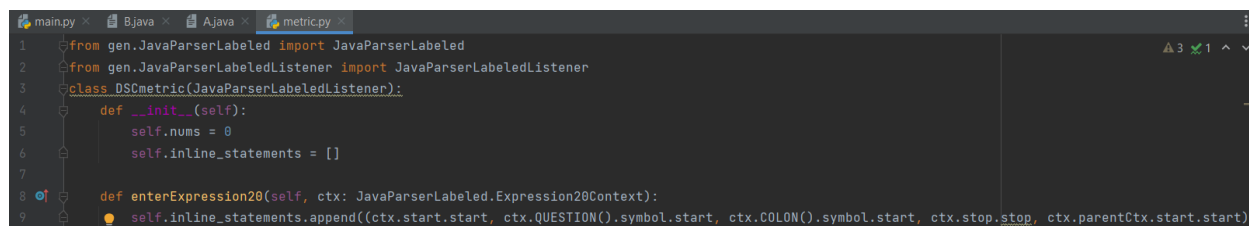
شکل زیر بخشی از درخت تجزیه است که از آن متوجه شدیم بعد از ASSIGN در expression21 expression0 باید داشته باشیم که آن expression20 باید با QUESTION و قسمت های مختلف آن با QUESTION و COLON قابل جداسازی هستند.

برای حل این بعد از ایجاد پروژه در همان فولدر پروژه دو فایل پایتون به نام های `metric` و `main` میسازیم. فایل متریک قرار است عملیاتی که سوال برای پیدا کردن خروجی میخواهد انجام بدهد.

برای این کار یک کلاس `DSCMetric` ایجاد کردیم که از `JavaParserLabeledListener` ارث بری میکند. یک متغیر از نوع لیست به نام `inline_statements` برای ذخیره ی عبارت های شرطی از نوع `inline` تعریف کردیم.

با توجه به درخت تجزیه دیدیم که باید تابع `enterExpression21` را `Override` کنیم.

در `enterExpression20` چک میکنیم که `parent` همان علامت مقداردهی باشد و بعد اجزای آن را به دست آورده و در لیست عبارت ها اضافه میکنیم.



```

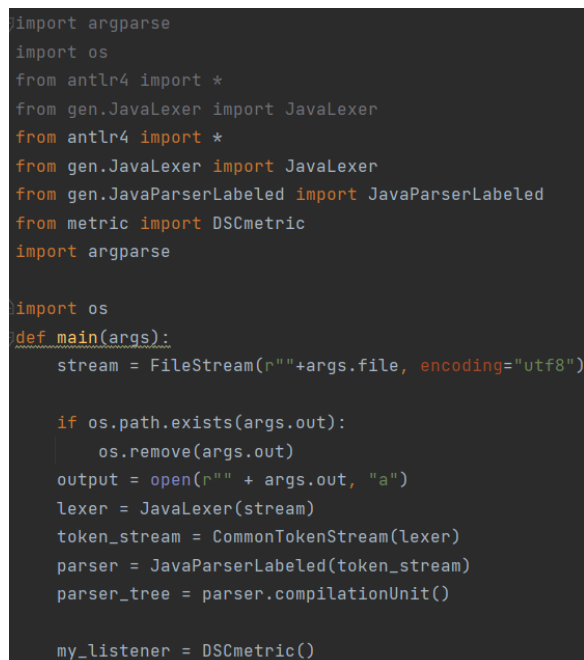
1 from gen.JavaParserLabeled import JavaParserLabeled
2 from gen.JavaParserLabeledListener import JavaParserLabeledListener
3 class DSCMetric(JavaParserLabeledListener):
4     def __init__(self):
5         self.nums = 0
6         self.inline_statements = []
7
8     def enterExpression20(self, ctx: JavaParserLabeled.Expression20Context):
9         self.inline_statements.append((ctx.start.start, ctx.QUESTION().symbol.start, ctx.COLON().symbol.start, ctx.stop.stop, ctx.parentCtx.start.start))

```

در فایل `main` هم یک تابع `main` داریم که در آن کلیت برنامه نوشته شده است.

استریم ورودی که همان فایل جاوای نوشته شده است خوانده میشود سپس به `JavaLexer` پاس داده میشود تا `tokenize` شود. سپس به `JavaParser` پاس داده میشود تا `ParseTree` ایجاد شود. سپس از روی کلاس متریک یک آبجکت به نام `listener` ساخته میشود که این آبجکت را به `walk` میدهم تا درخت ساخته شده را پیمایش کند و موارد خواسته شده را برای ما پیدا کند و با `if` و `else` معمولی بازنویسی کند و در خروجی ذخیره نماید.

فایل `main.py`:



```

import argparse
import os
from antlr4 import *
from gen.JavaLexer import JavaLexer
from antlr4 import *
from gen.JavaLexer import JavaLexer
from gen.JavaParserLabeled import JavaParserLabeled
from metric import DSCMetric
import argparse

import os
def main(args):
    stream = FileStream(r"+args.file, encoding="utf8")

    if os.path.exists(args.out):
        os.remove(args.out)
    output = open(r" + args.out, "a")
    lexer = JavaLexer(stream)
    token_stream = CommonTokenStream(lexer)
    parser = JavaParserLabeled(token_stream)
    parser_tree = parser.compilationUnit()

    my_listener = DSCMetric()

```

```

walker = ParseTreeWalker()
walker.walk(t=parser_tree, listener=my_listener)

statements = my_listener.inline_statements

text = stream.strdata
out = text
index = 0
for statement in statements:
    s, q, c, e, pre = statement
    if_else_statement = f"""if ({text[s:q]})
        {{
            {text[pre:s]}{text[q + 1:c]};
        }}
    else
        {{
            {text[pre:s]}{text[c + 1:e + 1]};
        }}"""
    out = out[:pre + index] + if_else_statement + out[e + 2 + index:]
    index += len(if_else_statement) - e + pre - 2

output.write(out)
output.close()

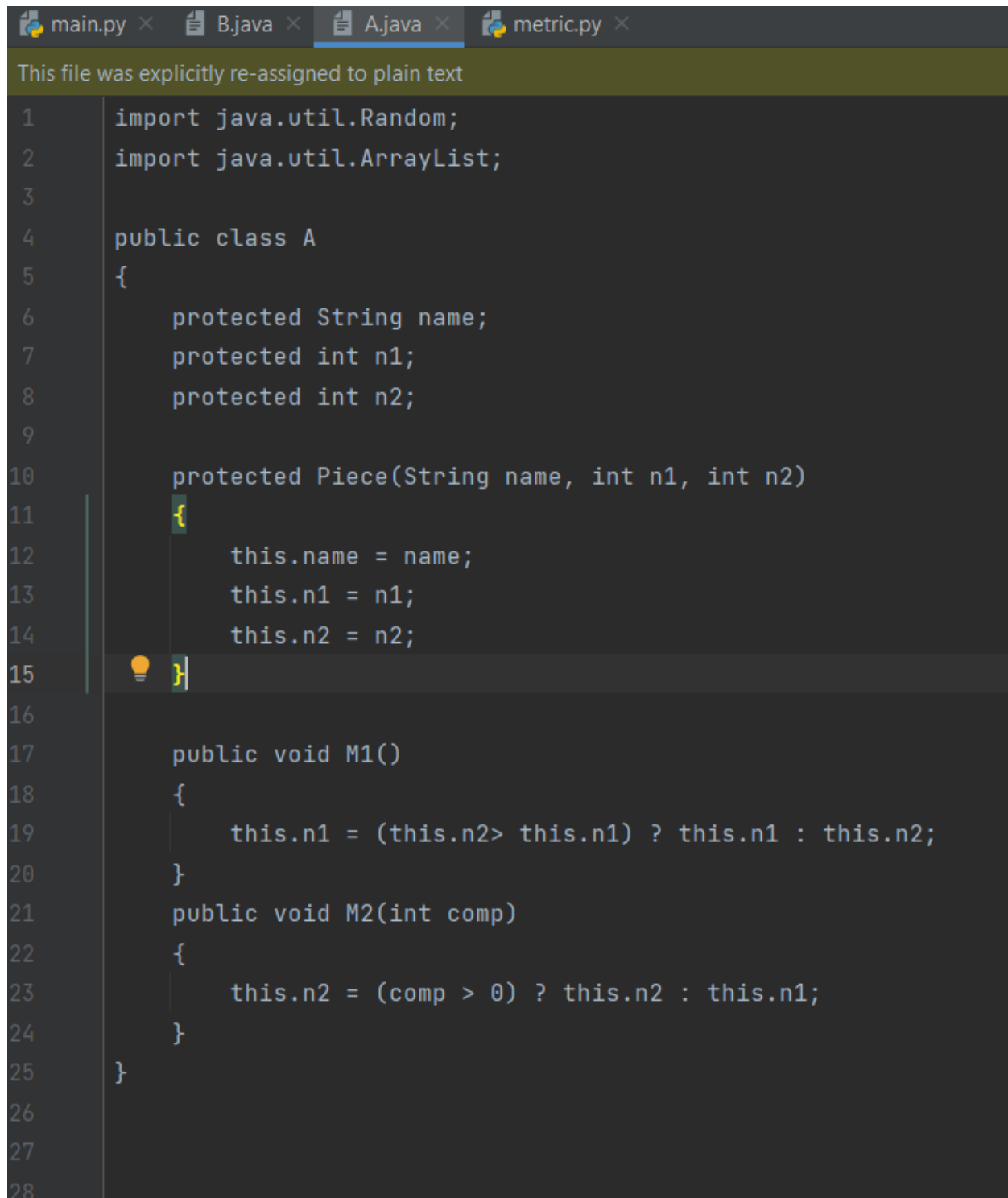
```

```

if __name__ == '__main__':
    argparser = argparse.ArgumentParser()
    argparser.add_argument('-n', '--file', type=str, help='name of Java file', default=r'A.java')
    args = argparser.parse_args()

    argparser.add_argument('-out', '--out', help='output java file path', default=r"B.java")
    args = argparser.parse_args()
    if args.file.split('.')[1] == 'java':
        main(args)

```



```
main.py × B.java × A.java × metric.py ×
This file was explicitly re-assigned to plain text
1  import java.util.Random;
2  import java.util.ArrayList;
3
4  public class A
5  {
6      protected String name;
7      protected int n1;
8      protected int n2;
9
10     protected Piece(String name, int n1, int n2)
11     {
12         this.name = name;
13         this.n1 = n1;
14         this.n2 = n2;
15     }
16
17     public void M1()
18     {
19         this.n1 = (this.n2 > this.n1) ? this.n1 : this.n2;
20     }
21     public void M2(int comp)
22     {
23         this.n2 = (comp > 0) ? this.n2 : this.n1;
24     }
25 }
26
27
28
```

خروجی برنامه:

```
import java.util.Random;
import java.util.ArrayList;

public class A
{
    protected String name;
    protected int n1;
    protected int n2;

    protected Piece(String name, int n1, int n2)
    {
        this.name = name;
        this.n1 = n1;
        this.n2 = n2;
    }
}
```

```
public void M1()
{
    if ((this.n2 > this.n1) )
    {
        this.n1 = this.n1 ;
    }
    else
    {
        this.n1 = this.n2;
    }
}

public void M2(int comp)
{
    if ((comp > 0) )
    {
        this.n2 = this.n2 ;
    }
    else
    {
        this.n2 = this.n1;
    }
}
}
```