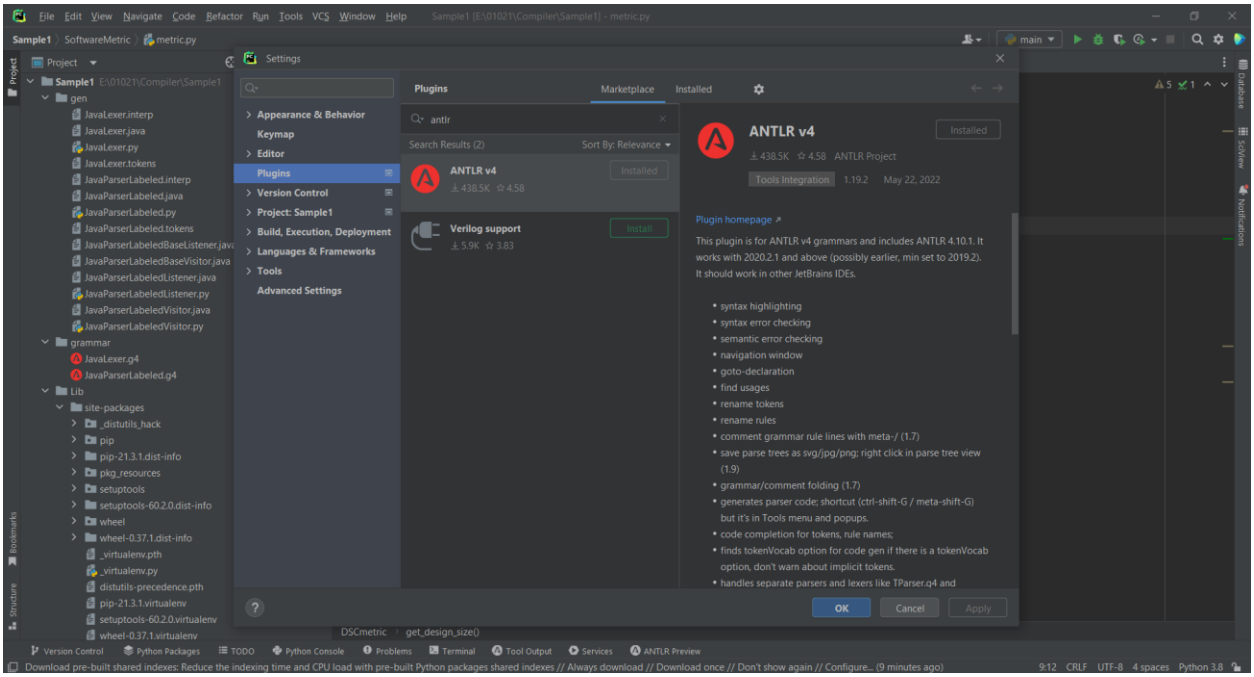
پاسخ تمرین اول

مهدیه نادری ۹۸۵۲۲۰۷۶

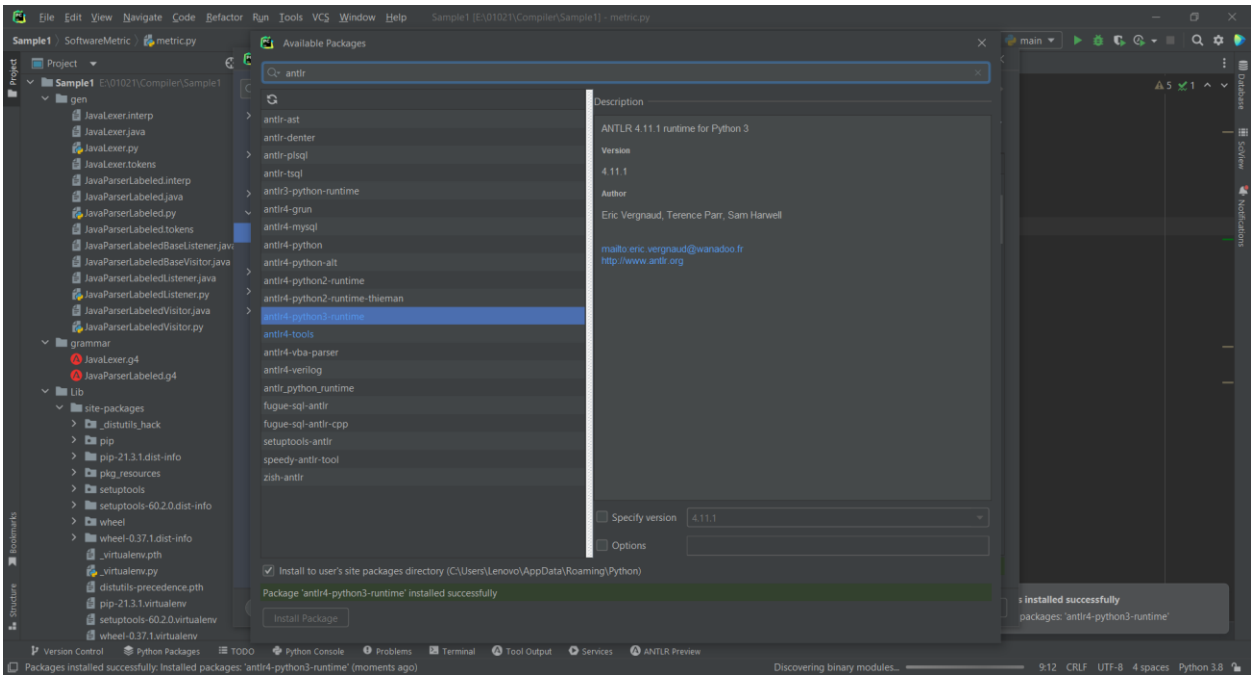
سوال اول:

مرحله ی اول نصب پلاگین انتلر روی پروژه:

روی settings کلیک میکنیم. سپس Plugins را انتخاب میکنیم در بین marketplace ها antlr را سرچ میکنیم و اگر قبلا نصب نکرده بودیم نصب میکنیم:

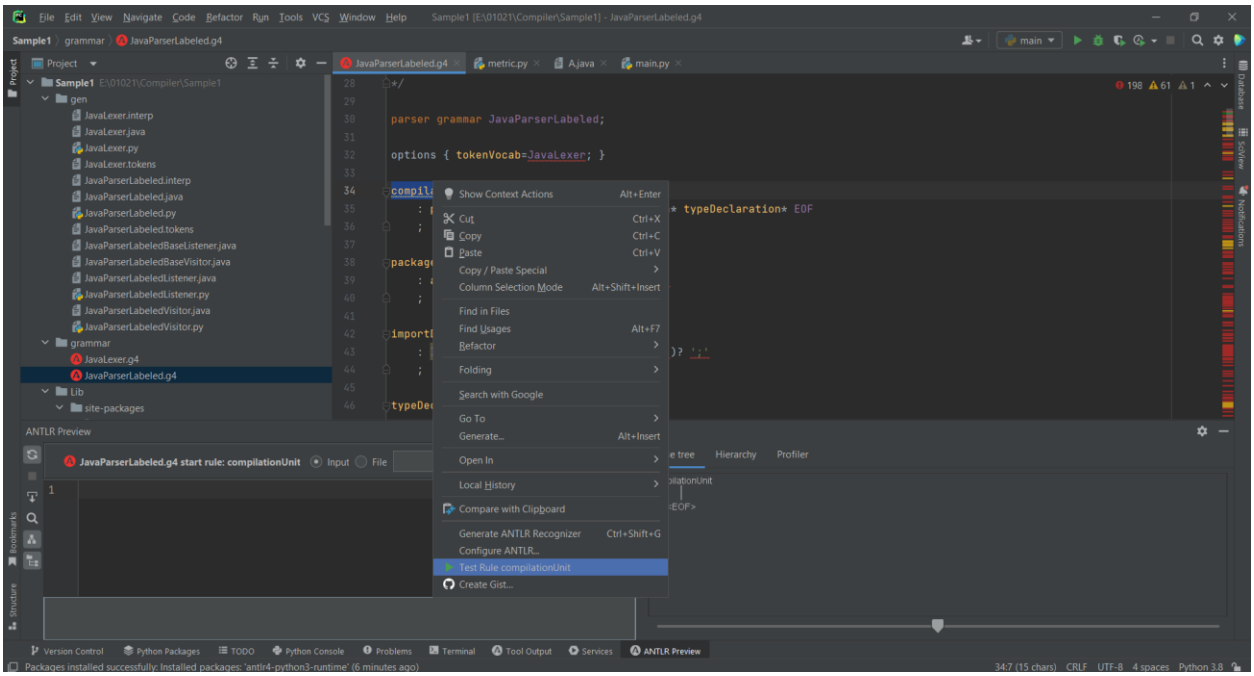


سپس در همان قسمت settings روی Project:Sample1 کلیک میکنیم و بعد Python Interpreter را انتخاب میکنیم علامت به علاوه را میزنیم و در پنجره ی باز شده antlr را جست و جو میکنیم و بعد antlr4-python3-runtime را انتخاب میکنیم و پکیج را نصب مینماییم:



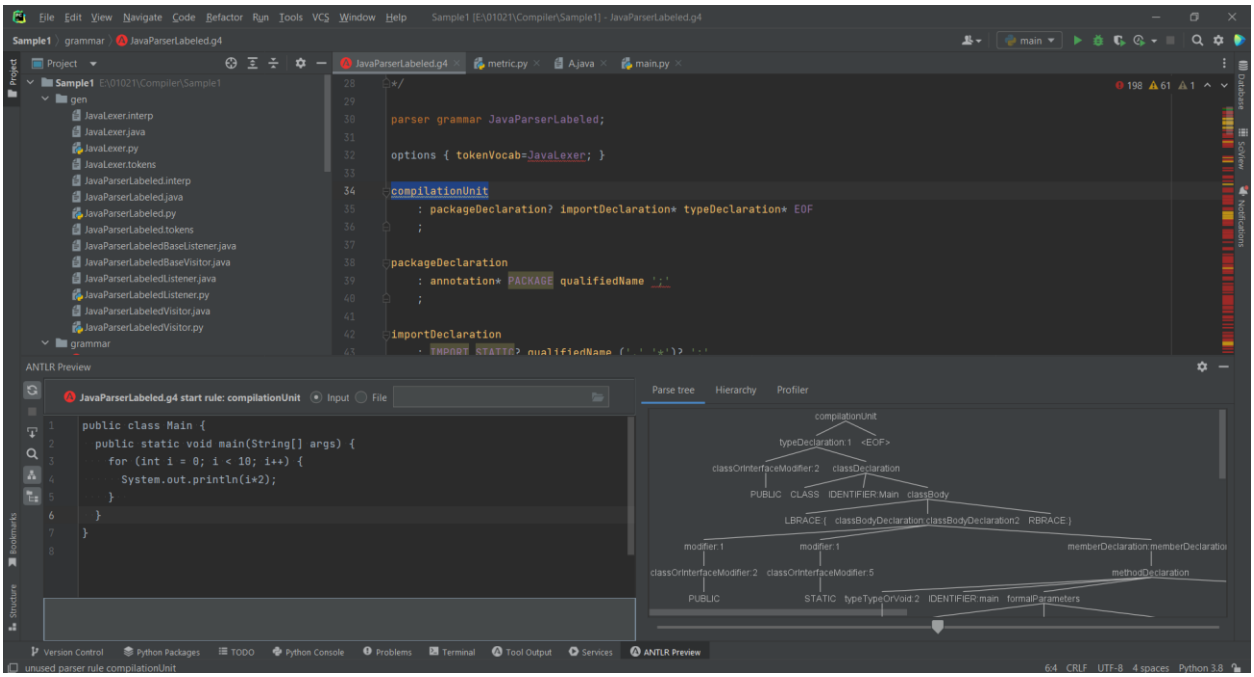
فایل های گرامر را که از قبل داشتیم در فولدر grammar قرار دادیم، سپس با کلیک راست روی فایل های گرامر و کلیک روی Generate ANTLR Recognization فایل های lexer, parser, listener, visitor را در فولدر gen تولید کردیم.

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 جاوارا بنویسیم و در سمت راست درخت تجزیه ی گرامر آن را مشاهده کنیم.

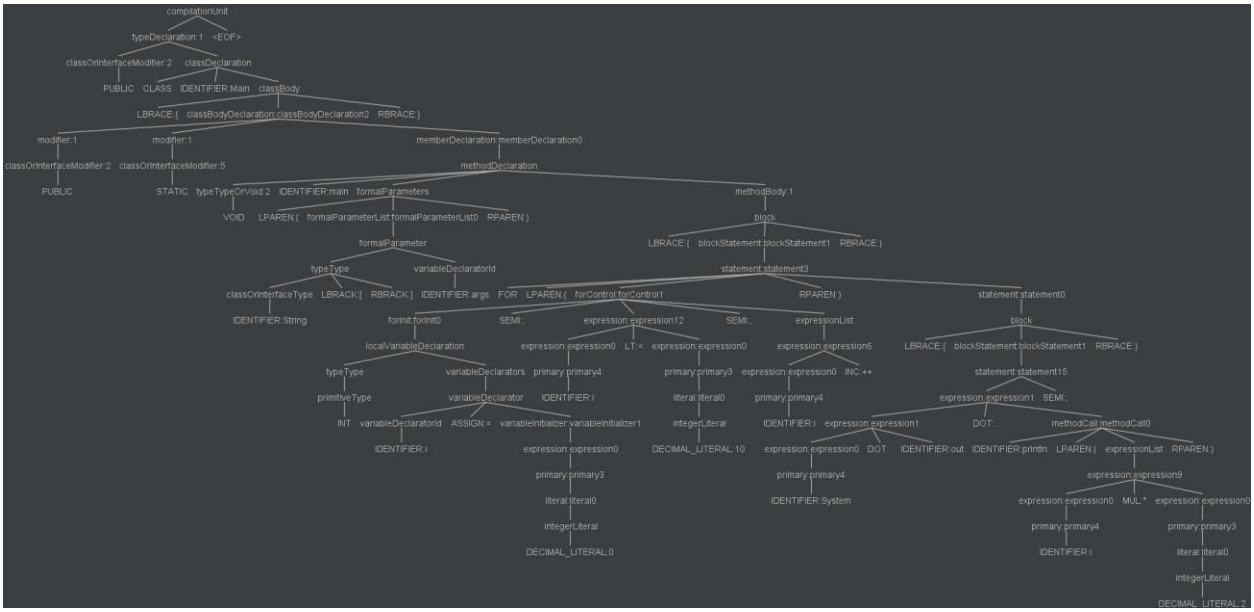


کد زیر را در قسمت پایین و سمت چپ وارد کردیم:

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      System.out.println(i*2);
    }
  }
}
```



با کلیک راست روی درخت و انتخاب گزینه ی Export to image تصویر را در پوشه ی Sample1 ذخیره نمودیم.



سوال دوم:

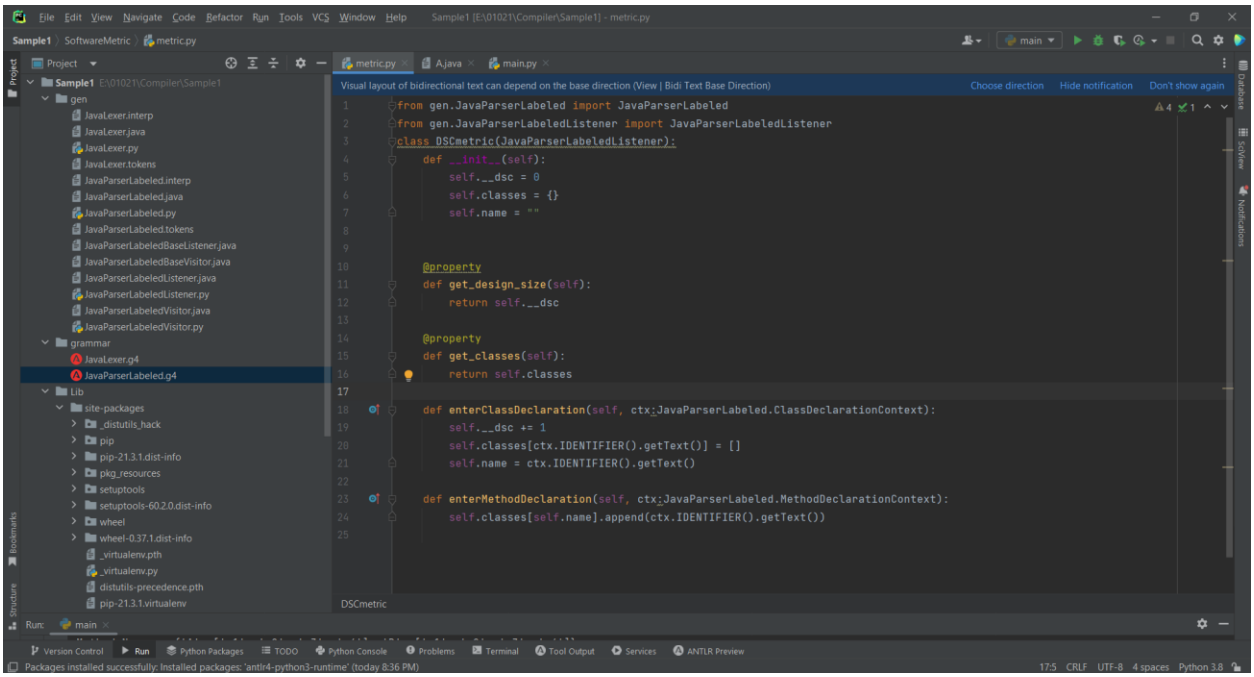
برای حل این سوال ابتدا یک پکیج پایتون جدید به نام SftwareMetric ایجاد میکنیم سپس در آن دو فایل پایتون به نام های metric و main میسازیم. فایل متریک قرار است عملیاتی که سوال برای پیدا کردن خروجی میخواهد انجام بدهد.

برای این کار یک کلاس DSCmetric ایجاد کردیم که از JavaParserLabeledListener ارث بری میکند. یک متغیر استرینگ برای نگهداری نام کلاسی که در لحظه بررسی میشود و یک دیکشنری برای ذخیره ی نام کلاس ها و متود های متناظر با آن ها تعریف میکنیم.

سپس برای اینکه بفهمیم کدام متود ها باید Override بشوند یک فایل جاوا میسازیم و در آن دو کلاس که هر کدام ۴ متود دارند تعریف میکنیم. سپس درخت تجزیه ی آن را رسم میکنیم. با تووجه به درخت میبینیم برای شناسایی کلاس ها باید classDeclaration و برای شناسایی متود ها methodDeclaration را Override کنیم چون از این قسمت به بعد در درخت جزئیات متود ها و کلاس ها در زیرشاخه ها آغاز میشود.

در classDeclaration به ازای هر کلاس یک لیست خالی در دیکشنری ایجاد میکنیم تا هر وقت در حال پیمایش متود ها بودیم نام آن ها در لیست اضافه شوند و همچنین نام کلاسی که در حال حاضر در حال پیمایش آن هستیم را ذخیره میکنیم تا هر وقت وارد متود ها شویم متود ها به عنوان متود مربوط به همین کلاس در لیست دیکشنری این کلاس قرار بگیرند.

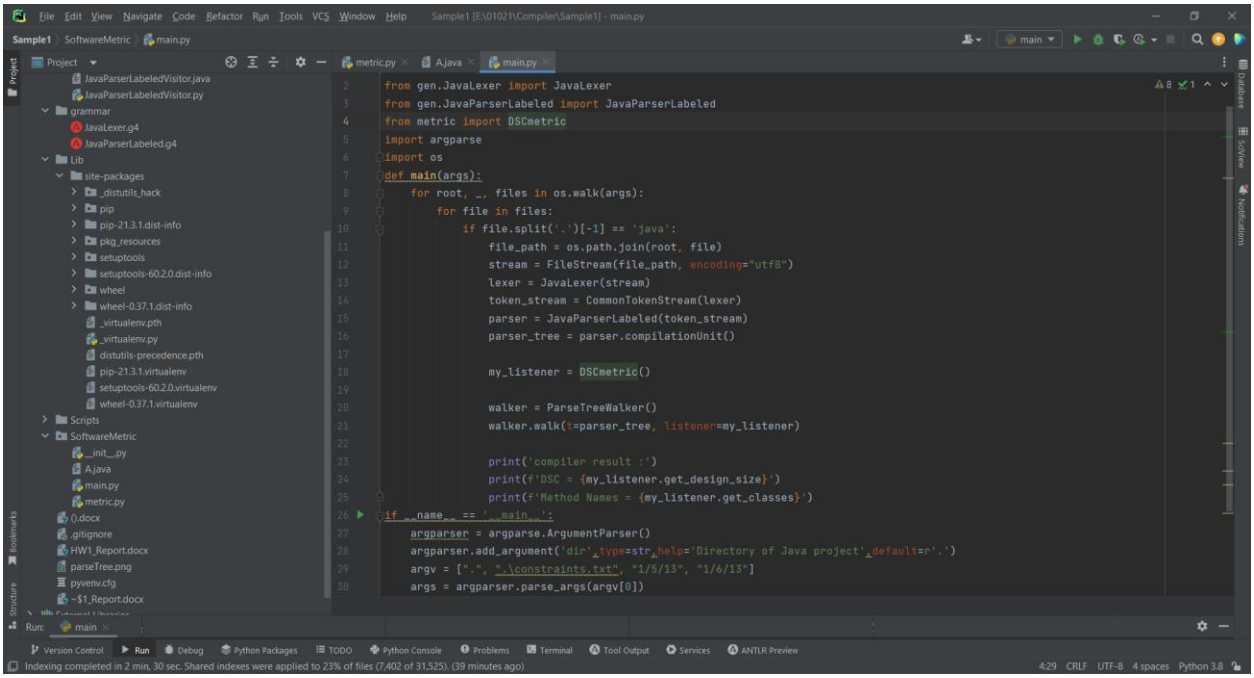
در methodDeclaration نام متودی که در حال پیمایش آن هستیم در لیست دیکشنری کلاسی که نامش را از قبل ذخیره کردیم اضافه خواهد شد.



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 میدهیم تا درخت ساخته شده را پیمایش کند و موارد خواسته شده را برای ما پیدا کند.

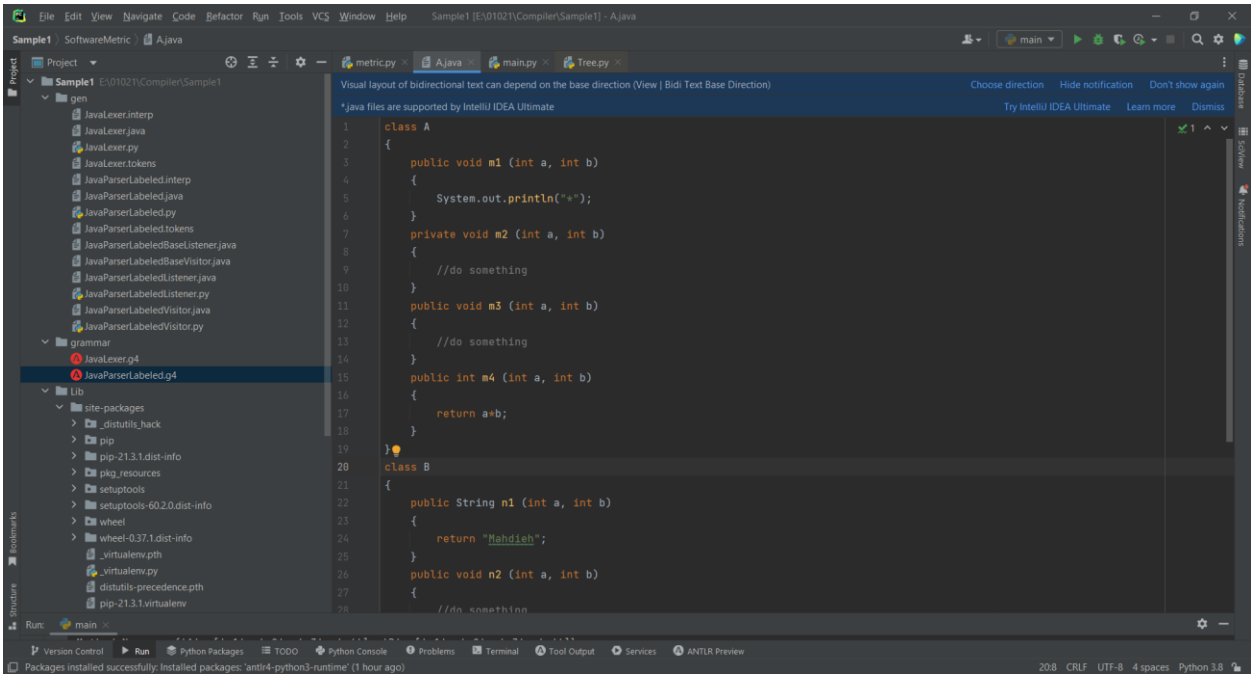
فایل main.py:



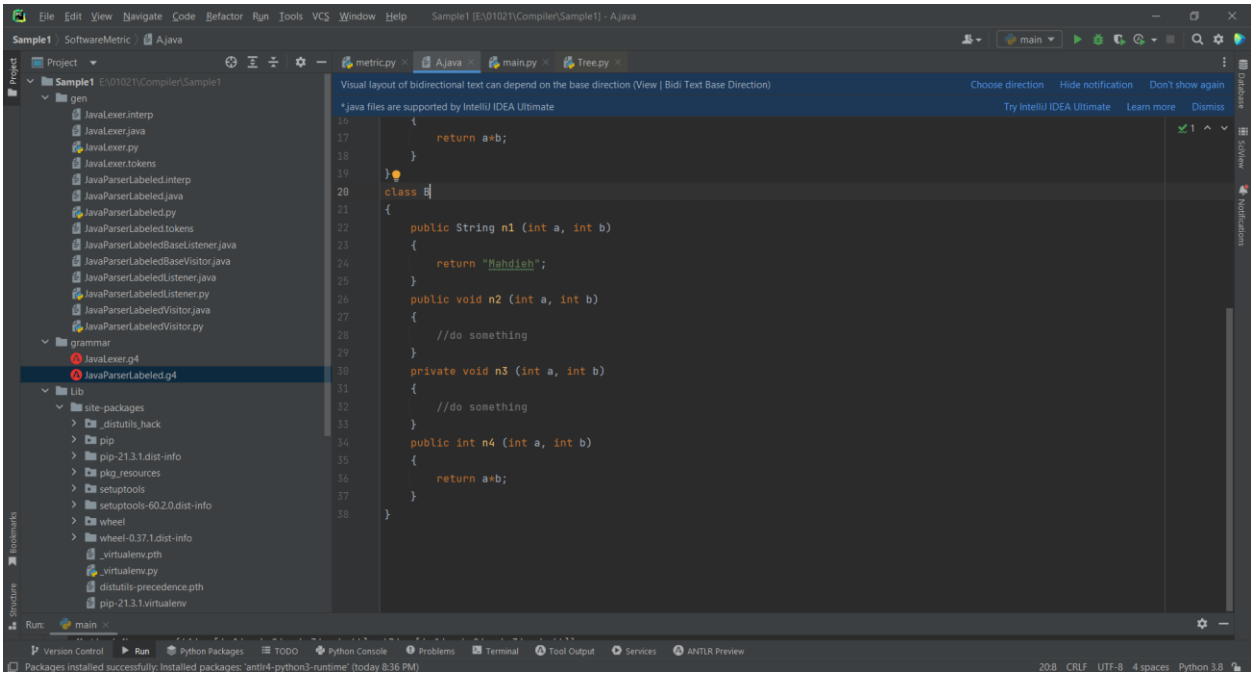
فایل A.java:

پاسخ تمرین اول

مهدیه نادری ۹۸۵۲۲۰۷۶



```
class A
{
    public void m1 (int a, int b)
    {
        System.out.println("*");
    }
    private void m2 (int a, int b)
    {
        //do something
    }
    public void m3 (int a, int b)
    {
        //do something
    }
    public int m4 (int a, int b)
    {
        return a*b;
    }
}
class B
{
    public String n1 (int a, int b)
    {
        return "Mahdieh";
    }
    public void n2 (int a, int b)
    {
        //do something
```



```
    {
        return a*b;
    }
}
class B
{
    public String n1 (int a, int b)
    {
        return "Mahdieh";
    }
    public void n2 (int a, int b)
    {
        //do something
    }
    private void n3 (int a, int b)
    {
        //do something
    }
    public int n4 (int a, int b)
    {
        return a*b;
    }
}
```

خروجی برنامه:

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python38\python.exe E:/01021/Compiler/Sample1/SoftwareMetric/main.py
ANTLR runtime and generated code versions disagree: 4.11.1!=4.10.1
ANTLR runtime and generated code versions disagree: 4.11.1!=4.10.1
compiler result :
DSC = 2
Method Names = {'A': ['m1', 'm2', 'm3', 'm4'], 'B': ['n1', 'n2', 'n3', 'n4']}

Process finished with exit code 0
```

پاسخ تمرین اول **مهدیه نادری ۹۸۵۲۲۰۷۶**

سوال سوم:

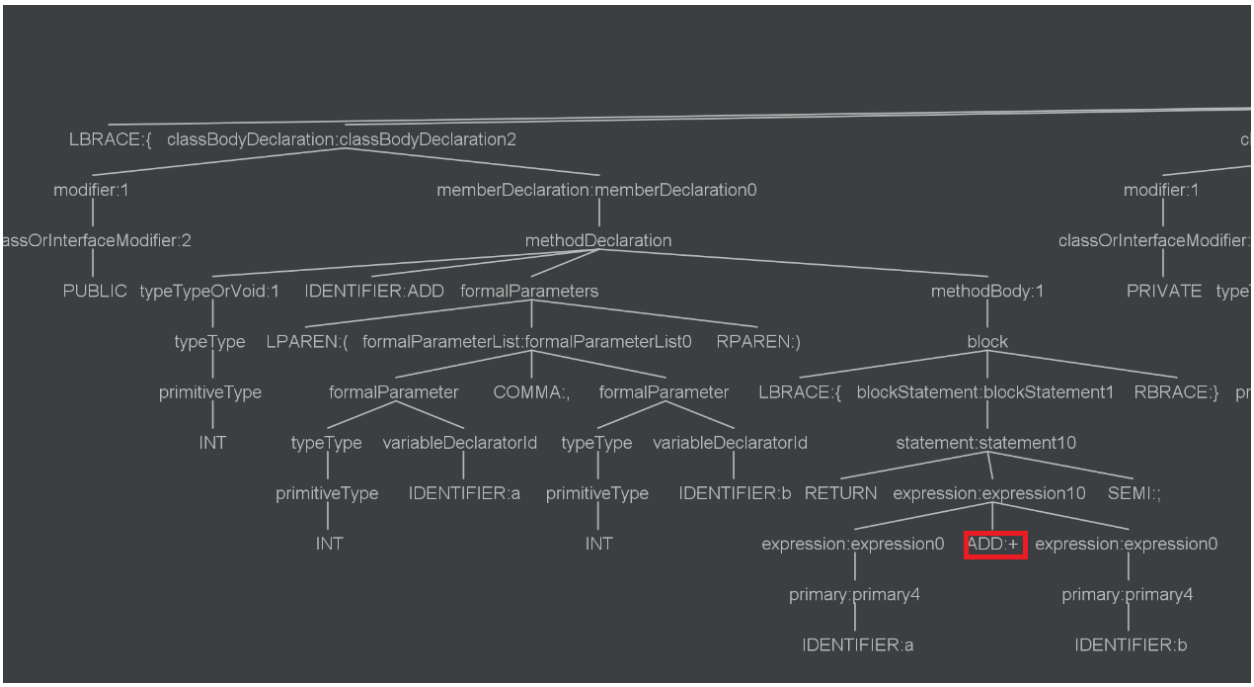
به طور کلی این سوال هم مانند سوال قبل است فقط باید اول درخت یک برنامه ی جاوا که شامل عملیات های ریاضی باشد را بکشیم تا ببینیم چه توابعی را نیاز است که Override کنیم و تفاوت دیگر هم این است که ورودی برنامه به جای آدرس برنامه نام فایل برنامه ی جاوا میباشد.

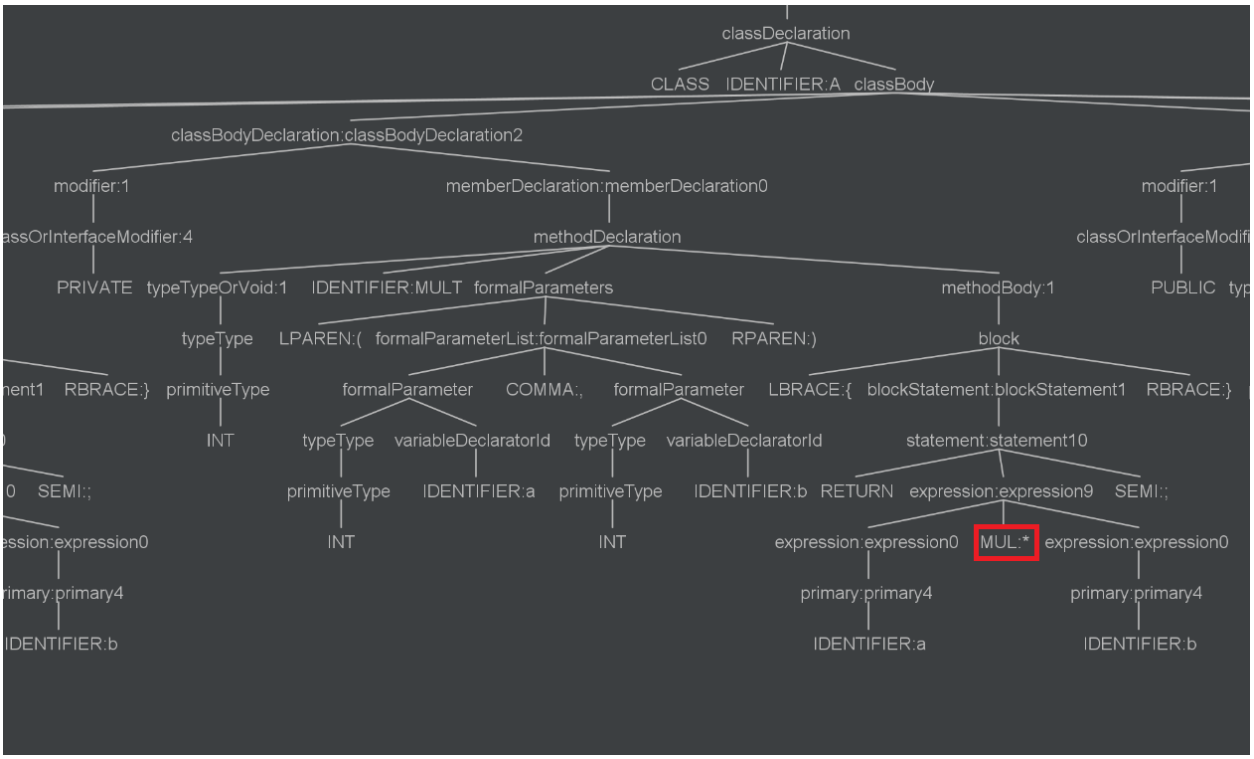
ابتدا یک پکیج پایتون دیگر به نام MathematicalOperation ساختم. سپس مانند سوال قبل دو فایل main.py و metric.py در آن ایجاد کردم.

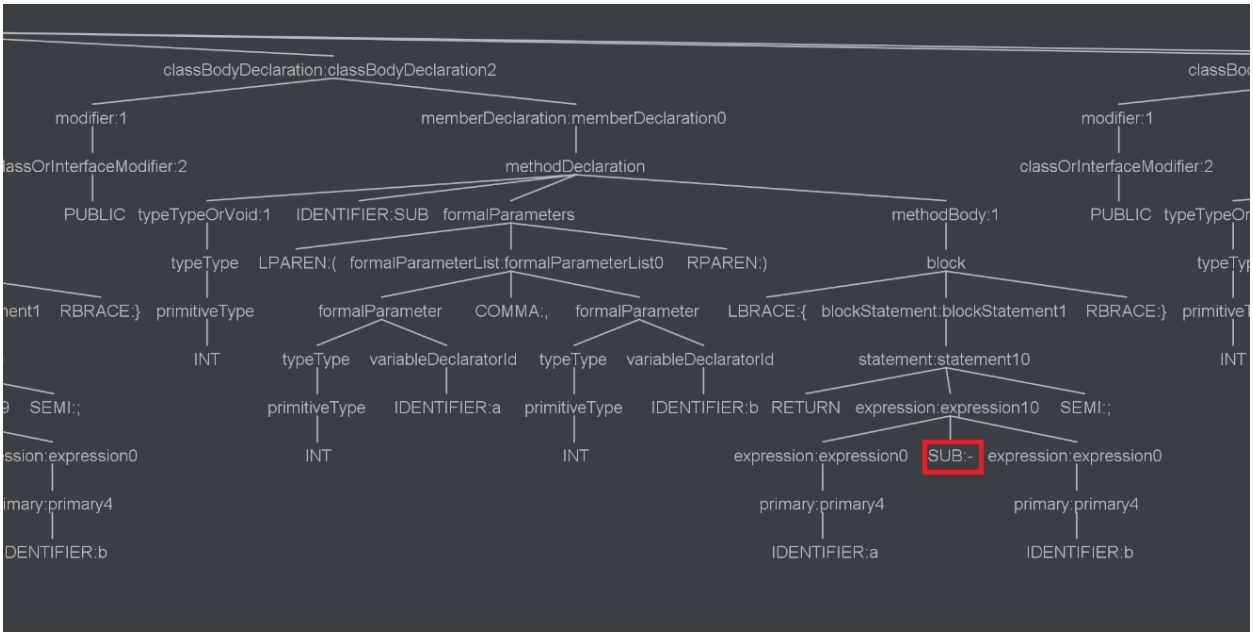
فایل جاوای زیر را در همان فولدر ذخیره کردم:

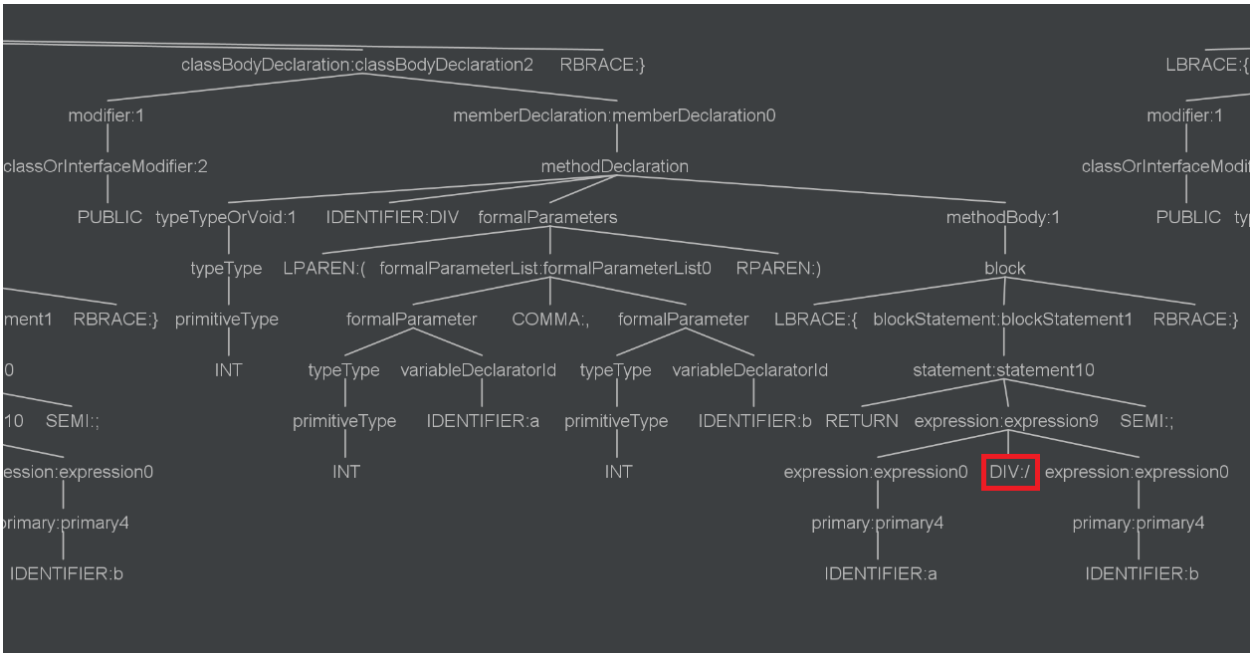
```
class A
{
    public int ADD (int a, int b)
    {
        return a+b;
    }
    private int MULT (int a, int b)
    {
        return a*b;
    }
    public int SUB (int a, int b)
    {
        return a-b;
    }
    public int DIV (int a, int b)
    {
        return a/b;
    }
}
class B
{
    public int DIST2 (int a, int b)
    {
        return a*a + b*b;
    }
    public int POW (int a, int b)
    {
        int result = a;
        for(int i = 0; i < b; i ++)
        {
            result = result * a;
        }
        return result;
    }
}
```

سپس درخت آن را رسم کردم و کلیدواژه هایی که در درخت عملیات های خواسته شده با آن شناخته میشوند را پیدا کردم:

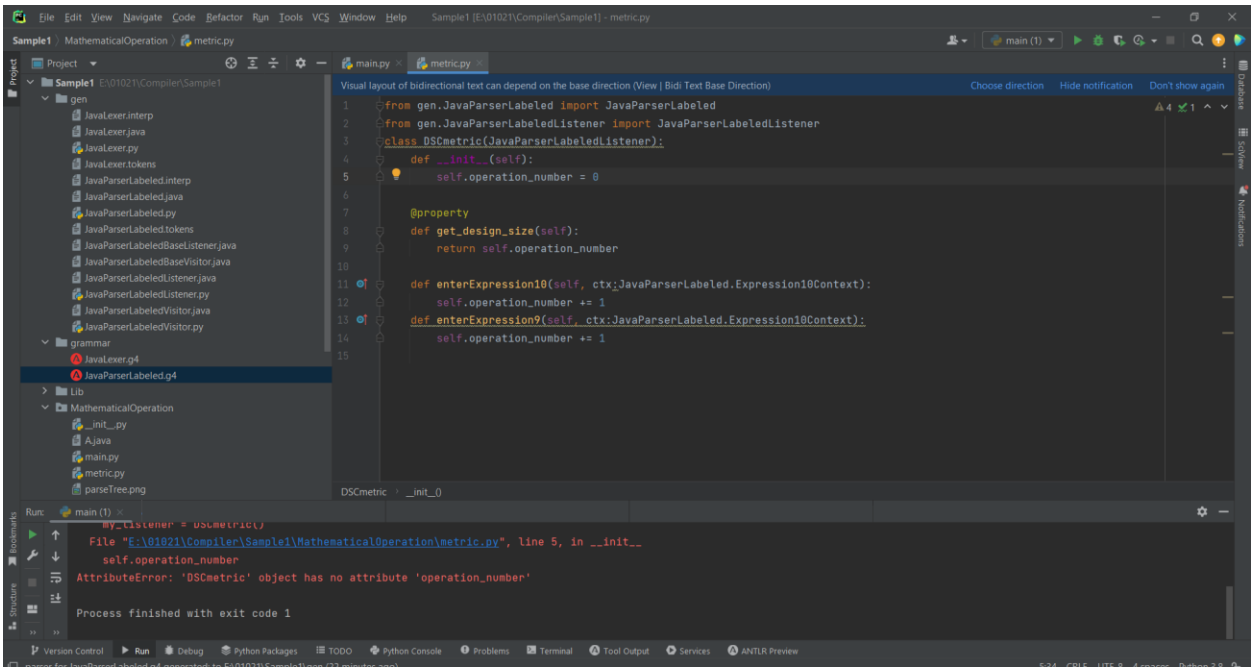

LBRACE:{ classBodyDeclaration:classBodyDeclaration2
modifier:1
memberDeclaration:memberDeclaration0
modifier:1
methodDeclaration
PUBLIC typeTypeOrVoid:1 IDENTIFIER:ADD formalParameters
methodBody:1
PRIVATE
typeType LPAREN:(formalParameterList:formalParameterList0 RPAREN:)
block
primitiveType
formalParameter COMMA:, formalParameter
LBRACE:{ blockStatement:blockStatement1 RBRACE:}
INT
typeType variableDeclaratorId typeType variableDeclaratorId
statement:statement10
primitiveType IDENTIFIER:a primitiveType IDENTIFIER:b RETURN expression:expression10 SEMI:;
INT
INT
expression:expression0 ADD:+ expression:expression0
primary:primary4
primary:primary4
IDENTIFIER:a
IDENTIFIER:b


classDeclaration
CLASS IDENTIFIER:A classBody
classBodyDeclaration:classBodyDeclaration2
modifier:1
memberDeclaration:memberDeclaration0
modifier:1
methodDeclaration
PRIVATE typeTypeOrVoid:1 IDENTIFIER:MULT formalParameters
methodBody:1
PUBLIC
typeType LPAREN:(formalParameterList:formalParameterList0 RPAREN:)
block
primitiveType
formalParameter COMMA:, formalParameter
LBRACE:{ blockStatement:blockStatement1 RBRACE:}
INT
typeType variableDeclaratorId typeType variableDeclaratorId
statement:statement10
primitiveType IDENTIFIER:a primitiveType IDENTIFIER:b RETURN expression:expression9 SEMI:;
INT
INT
expression:expression0 MUL:* expression:expression0
primary:primary4
primary:primary4
IDENTIFIER:a
IDENTIFIER:b

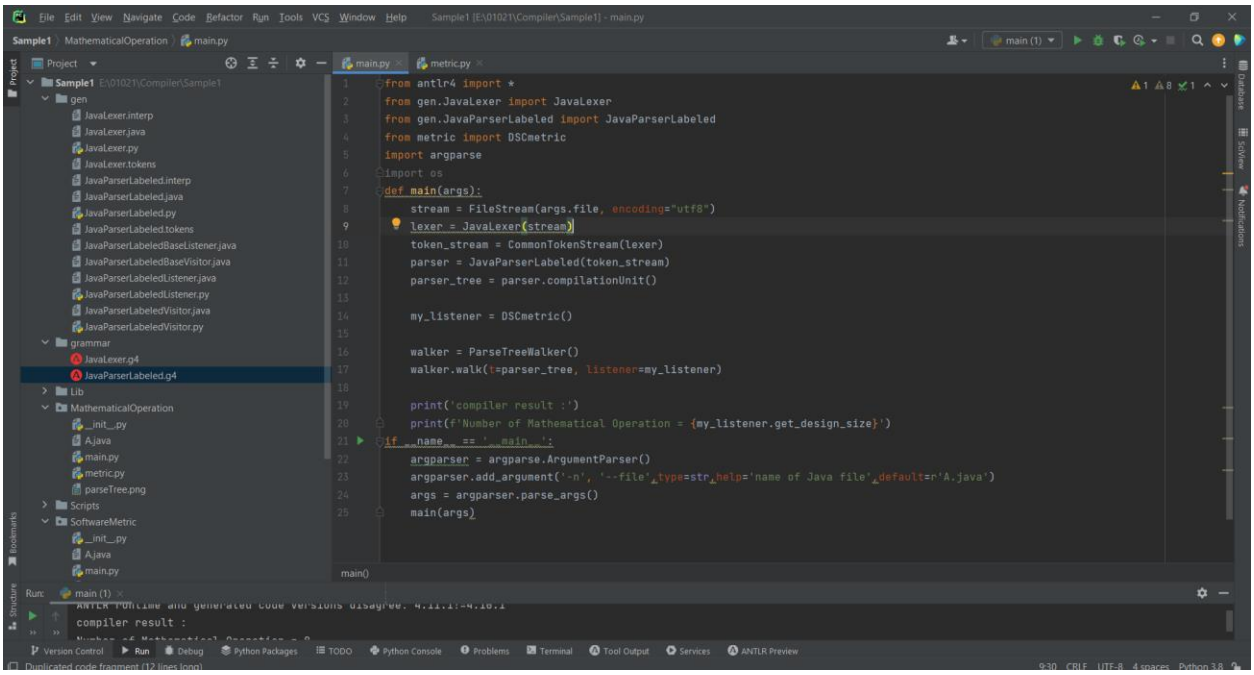




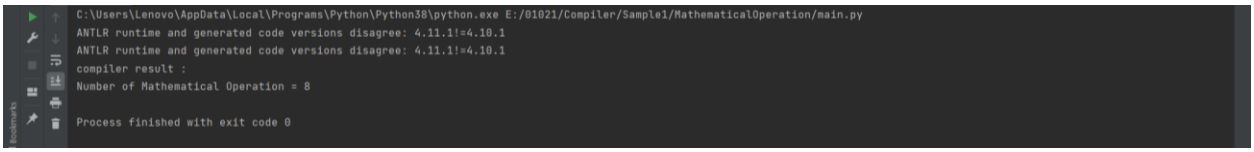
همان طور که در تصاویر مشخص کردم ADD برای جمع، SUB برای تفریق، MUL برای ضرب و DIV برای تقسیم استفاده شده است. که MUL و DIVاز نوع expression9هستند و ADD و SUB ازنوع expression10 هستند برای همین این دو تابع را باید بازنویسی کنیم طوری که هر بار وارد توابع مربوط به اینها میشود شمارنده یک واحد اضافه شود. حالا در فایل metric.py کلاس DSCmetric که از JavaParserLabeledListener ارث بری میکند را میسازیم و بعد از ساخت کانستراکتور تابع هایی که مورد نیازمان است Override میکنیم.



سپس تابع main را مانند زیر مینویسیم:



خروجی کد به صورت زیر خواهد بود:



من هم در فایل جاوا از ۸ عملیات ریاضی در توابع و کلاس ها استفاده کرده بودم که خروجی کد درست است.

پاسخ تمرین اول **مهدیه نادری ۹۸۵۲۲۰۷۶**

سوال چهارم:

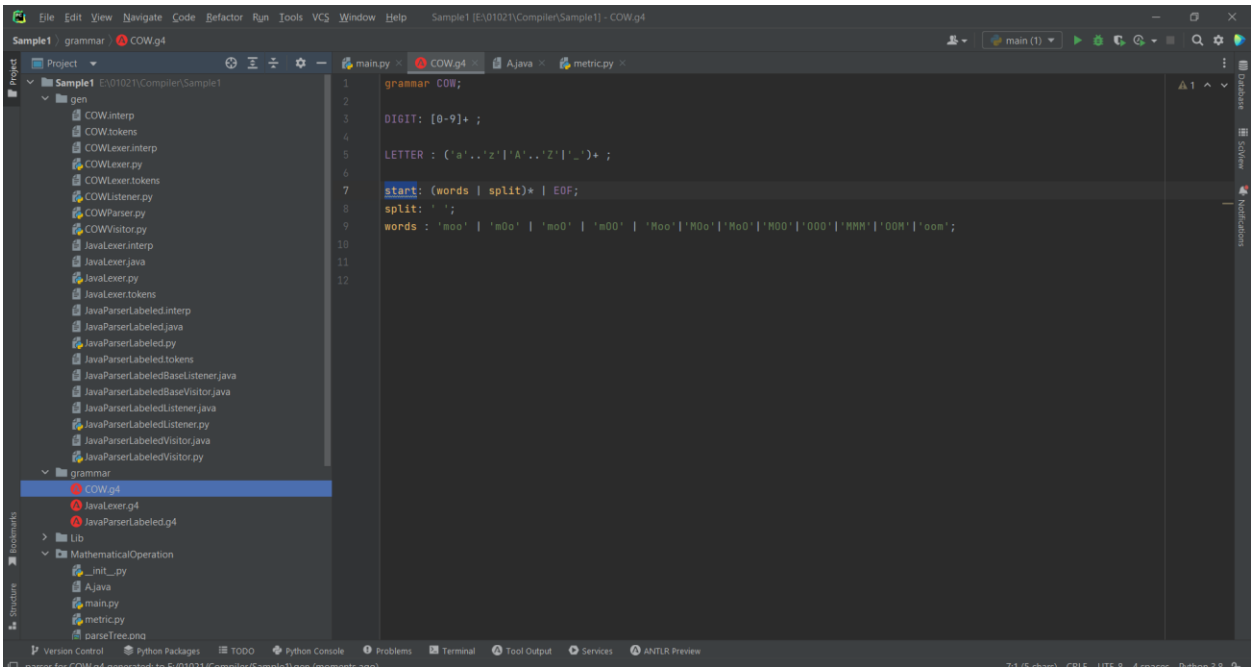
برای حل این سوال در فولدر grammar یک فایل به نام COW.g4 ایجاد کردم.

سپس باید قواعد این زبان را بفهمیم و بعد به زبان ماشین آن را بنویسیم.

زبان برنامه نویسی COW یک زبان برنامه نویسی باطنی است که توسط شان هیبر در سال 2003 ایجاد شد. این یک نوع Brainfuck است که به صورت طنز با در نظر گرفتن Bovinae طراحی شده است. COW دوازده دستورالعمل دارد (چهار دستورالعمل بیشتر از Brainfuck) و تورینگ کامل است. بیشتر دستورالعمل‌ها moos هستند، فقط حروف بزرگ متفاوت است: moOo، moO، mOO، Moo، و غیره. MMM، OOO، oom و OOM استثنا هستند. سایر ترکیبات شخصیت نادیده گرفته می شوند و به عنوان نظر تلقی می شوند.

دستورالعمل	توضیحات
moo	این دستور به دستور MOO متصل است. هنگامی که در طول اجرای عادی با آن مواجه می شود، کد برنامه را به صورت معکوس به دنبال یک دستور MOO منطبق جستجو می کند و با شروع از دستور MOO یافت شده، دوباره اجرا را آغاز می کند. هنگام جستجو، دستورالعملی را که بلافاصله قبل از آن است رد می کند (به MOO مراجعه کنید).
mOo	موقعیت فعلی حافظه را یک بلوک به عقب می برد.
moO	موقعیت فعلی حافظه را یک بلوک به جلو می برد.
mOO	مقدار را در بلوک حافظه فعلی طوری اجرا کنید که انگار یک دستورالعمل است. دستور اجرا شده بر اساس مقدار کد دستورالعمل است (به عنوان مثال، اگر بلوک حافظه فعلی حاوی 2 باشد، دستور moO اجرا می شود). یک دستور نامعتبر از برنامه در حال اجرا خارج می شود. مقدار 3 نامعتبر است زیرا باعث ایجاد یک حلقه بی نهایت می شود.
Moo	اگر بلوک حافظه فعلی دارای 0 باشد، یک نویسه ASCII را از STDIN بخوانید و آن را در بلوک حافظه فعلی ذخیره کنید. اگر بلوک حافظه فعلی 0 نیست، کاراکتر ASCII مربوط به مقدار بلوک حافظه فعلی را در STDOUT چاپ کنید.
MOo	مقدار بلوک حافظه فعلی را 1 کاهش دهید.
MoO	مقدار بلوک حافظه فعلی را 1 افزایش دهید.
MOO	اگر مقدار بلوک حافظه فعلی 0 است، دستور بعدی را رد کنید و پس از دستور moo مطابق بعدی، اجرا را از سر بگیرید. اگر مقدار بلوک حافظه فعلی 0 نیست، با دستور بعدی ادامه دهید. توجه داشته باشید که این واقعیت که دستور را بلافاصله پس از آن نادیده می‌گیرد، پیامدهای جالبی برای اینکه دستور تطبیق moo واقعاً کجاست، دارد. برای مثال، موارد زیر با موو دوم و نه اول مطابقت دارند: OOO MOO moo moo
OOO	مقدار بلوک حافظه فعلی را روی 0 تنظیم کنید.
MMM	اگر مقدار فعلی در ثبات وجود ندارد، مقدار بلوک حافظه فعلی را کپی کنید. اگر مقداری در رجیستر وجود دارد، آن مقدار را در بلوک حافظه فعلی قرار دهید و رجیستر را پاک کنید.
OOM	مقدار بلوک حافظه فعلی را به عنوان یک عدد صحیح در STDOUT چاپ کنید.
oom	یک عدد صحیح از STDIN را بخوانید و آن را در بلوک حافظه فعلی قرار دهید.

فایل گرامر را به صورت زیر نوشتیم:



کد زیر را به عنوان نمونه میدهیم تا درخت آن را بسازد.