

به نام خدا

پروژه پایانی درس کامپایلر

1400 - 1401



گروه هفتم

اعضای گروه:

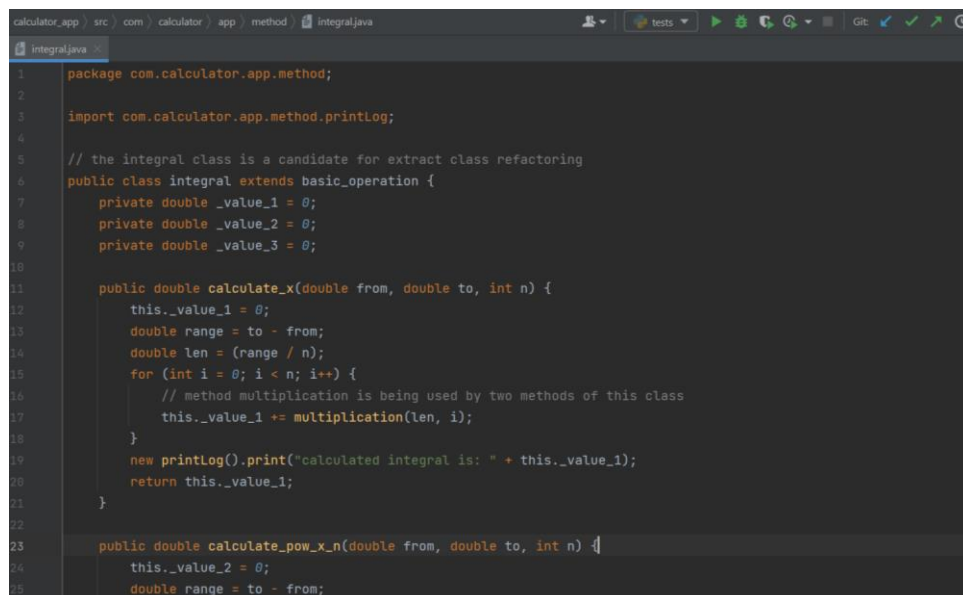
ملیکا نوبختیان

زهرا محمودزاده

غزل زمانی نژاد

1) import - importby

با استفاده از این رفرنس می‌توانیم از کلاس‌های یک پکیج استفاده کنیم. یک نمونه از این رفرنس را در تصویر زیر می‌بینید.



```
1 package com.calculator.app.method;
2
3 import com.calculator.app.method.printLog;
4
5 // the integral class is a candidate for extract class refactoring
6 public class integral extends basic_operation {
7     private double _value_1 = 0;
8     private double _value_2 = 0;
9     private double _value_3 = 0;
10
11     public double calculate_x(double from, double to, int n) {
12         this._value_1 = 0;
13         double range = to - from;
14         double len = (range / n);
15         for (int i = 0; i < n; i++) {
16             // method multiplication is being used by two methods of this class
17             this._value_1 += multiplication(len, i);
18         }
19         new printLog().print("calculated integral is: " + this._value_1);
20         return this._value_1;
21     }
22
23     public double calculate_pow_x_n(double from, double to, int n) {
24         this._value_2 = 0;
25         double range = to - from;
```

در این مثال، entity آن، printLog است. ویژگی‌های این entity را به کمک understand چاپ کردیم. در شکل زیر نشان داده شده است:



```
Import printLog integral.java(3)
ent: printLog value: None type: None ent.kind: Public Class ////////// parent: printLog.java ////////// name: printLog ////////// longname: com.ca
@Override
public void print(String text) {
    super.print(text);
}
```

ابتدا فایل‌های موجود در calculate_app را می‌خوانیم. نام و آدرس فایل‌های جاوا را ذخیره می‌کنیم. برای همه‌ی فایل‌های جاوا در یک for موارد زیر را انجام می‌دهیم. برای این فایل، lexer و parser ایجاد می‌کنیم. سپس ImportListener را صدا می‌کنیم. در importListener ابتدا longname آن را با ctx.qualifiedName().getText() پیدا می‌کنیم و در یک لیست ذخیره می‌کنیم. آخرین عضو آن را به عنوان name ذخیره می‌کنیم. در لیست is_unknown_class ذخیره می‌کنیم که کلاس import شده برای جاوا است یا نه. اگر برای جاوا باشد، parent آن را None می‌گذاریم. اگر این طور نباشد، parent آن، فایل می‌شود. Parent ها را در لیست parents ذخیره می‌کنیم. Line و column را هم با استفاده از ctx.children[0].symbol.column و ctx.children[0].symbol.line در همین listener پیدا می‌کنیم.

```

29 def enterImportDeclaration(self, ctx: JavaParserLabeled.CompilationUnitContext):
30     # print(ctx.qualifiedName().getText())
31     longname = ctx.qualifiedName().getText()
32     self.longnames.append(longname)
33     name = longname.split('.')[-1]
34     self.names.append(name)
35
36     if longname.split('.')[-1] != 'java':
37         self.is_unknown_class.append(True)
38         parent = None
39     else:
40         self.is_unknown_class.append(False)
41         parent = name + ".java"
42
43     self.parents.append(parent)
44
45     self.line = ctx.children[0].symbol.line
46     self.col = ctx.children[0].symbol.column
47
48

```

بعد از صدا کردن listener در تابع readFile در یک for برای هر entity بررسی میکنیم که kind آن چه نوعی است. اگر unknown باشد، content آن یک رشته خالی است. اگر public class باشد، محتوای کلاس را از تابع get_class_body میگیرد.

```

120 # find entity
121 entities = []
122 for ent_name, longname, is_unk, parent in zip(listener.names, listener.longnames, listener.is_unknown_class, listener.parents):
123     if is_unk:
124         ent_kind = KindModel.get_or_none(_name="Java Unknown Class Type Member")._id
125         contents = ""
126         obj = create_Entity(ent_name, longname, None, contents, ent_kind, None, None)
127
128     else:
129         ent_kind = KindModel.get_or_none(_name="Java Class Type Public Member")._id
130         idx = filename.index(parent)
131         new_path = listOfFiles[idx]
132         contents = get_class_body(new_path)
133
134         # create parent of entity of ref
135         parent_contents = FileStream(new_path)
136         parent_obj = create_Entity(parent, longname, None, parent_contents, file_kind, None, None)
137         obj = create_Entity(ent_name, longname, parent_obj._id, contents, ent_kind, None, None)
138
139     entities.append(obj._id)
140
141

```

در تابع get_class_body هم یک lexer و parser ساختیم و کلاس مورد نظر را پیدا کردیم. مقدار value و type هم برای این entity ها None است. این مقادیر را به توابع create_ref و create_Entity میدهیم و entity ها در دیتابیس ذخیره میشوند. برای ساخت scope هم از همین تابع استفاده کردیم. برای ساختن reference ها از تابع create_ref استفاده کردیم. مقدار kind رفرنس را Java Import قرار دادیم. همه ی مقادیر entity و scope و kind و line و column را به create_ref دادیم و reference ها در دیتابیس ساخته شد.

2) define - defineby

این رفرنس به ما کمک می کند تا هر جا یک variable تعریف شد بتوانیم از آن استفاده کنیم. ابتدا به توضیح یک نمونه از این رفرنس و سایر موارد مربوط به آن می پردازیم:

```
// the integral class is a candidate for extract class refactoring
public class integral extends basic_operation {
    private double _value_1 = 0;
    private double _value_2 = 0;
    private double _value_3 = 0;

    public double calculate_x(double from, double to, int n) {
        this._value_1 = 0;
        double range = to - from;
        double len = (range / n);
        for (int i = 0; i < n; i++) {
            // method multiplication is being used by two methods of this class
            this._value_1 += multiplication(len, i);
        }
        new printLog().print("calculated integral is: " + this._value_1);
        return this._value_1;
    }
}
```

در شکل بالا متغیر len را می بینیم که در متد calculate_x تعریف شده است. ابتدا ویژگی entity آن را بررسی می کنیم. Name در entity ما برابر اسم متغیر ما یعنی همان len است. Type آن هم همان طور که در تصویر می بینید double است. این متغیر هیچ نوع خاصی ندارد یعنی برای مثال private نیست پس kind آن Java Variable Public Member خواهد بود. Value در اینجا همان مقدار جلوی مساوی خواهد بود که برابر (range/n) است. یک متغیر content خاصی ندارد. برای پیدا کردن parent لازم است که ببینیم متغیر ما در کدام کلاس یا متد تعریف شده است. در اینجا این اتفاق در calculate_x افتاده است پس parent ما خواهد بود. حالا به سراغ scope در رفرنس که خودش نوعی entity است می رویم. در اینجا scope برابر همان parent خواهد بود.

حالا به توضیح کد آن خواهیم پرداخت. ابتدا به توضیح DefineListener می پردازیم. متد اصلی این کلاس در اینجا enterVariableDeclarators خواهد بود زیرا هر جا که به تعریف یک متغیر برسیم وارد این متد خواهیم شد.

در ابتدا متد find parent را صدا خواهیم زد این متد به ما کمک می کند تا جایی که متغیر ما تعریف شده است را پیدا کنیم و مقادیر مربوط به parent entity را به دست آوریم:

```
def find_parent(self, Ctx):
    current = Ctx.parentCtx
    while current is not None:
        if type(current).__name__ == "ClassDeclarationContext" or type(
            current).__name__ == "MethodDeclarationContext":
            self.parents.append(current.IDENTIFIER().getText())
            if type(current).__name__ == "ClassDeclarationContext":
                self.parent_info.append(self.make_parent_class(current))
            elif type(current).__name__ == "MethodDeclarationContext":
                self.make_parent_method(current)
            return
        current = current.parentCtx
    self.parents.append(" ")
```

تعریف این متد را در تصویر بالا می بینید. ما مرحله به مرحله parent هر context را مشاهده می کنیم تا به تعریف یک کلاس یا متد برسیم در این صورت به parent رسیده ایم و حالا با توجه به نوع آن باید متغیرهای مرتبط با آن را پیدا کنیم. اگر parent ما از نوع کلاس باشد وارد متد make_parent_class شد:

```
def make_parent_class(self, current):
    content = current.getText()
    name = current.IDENTIFIER().getText()
    kind = KindModel.get_or_none(_name="Java Class Type Public Member")._id
    parent = "file"
    return {"parent": parent, "kind": kind, "name": name, "content": content}
```

در این کلاس ویژگی های parent entity از نوع class را به دست می آوریم. Content آن کلاس برابر تعریف کلی همان کلاس خواهد بود. Name آن هم از دسترسی به IDENTIFIER در context کنونی به دست می آید. در اینجا kind هم از نوعی که در تصویر است خواهد بود. Parent خود کلاس را فعلا با file مشخص می کنیم تا در قدم های بعدی entity فایلی که در آن تعریف شده است را بسازیم. این اطلاعات را بر خواهیم گرداند و در parent info ذخیره خواهیم کرد.

ولی اگر parent از نوع متد باشد از make_parent_method استفاده می کنیم:

```

def make_parent_method(self, current):
    content = current.getText()
    name = current.IDENTIFIER().getText()
    if type(current).__name__ == "MethodDeclarationContext":
        if "static" in current.getText():
            kind = KindModel.get_or_none(_name="Java Static Method Public Member")._id
        else:
            kind = KindModel.get_or_none(_name="Java Method Public Member")._id
    current = current.parentCtx
    while current is not None:
        if type(current).__name__ == "ClassDeclarationContext":
            parent = self.make_parent_class(current)
            break
        current = current.parentCtx
    self.parent_info.append({"parent": parent, "kind": kind, "name": name, "content": cor

```

بیشتر اطلاعات متد شبیه کلاس خواهد بود. Kind آن می تواند بسته به static بودن یا نبودن متد تغییر کند و مطابق تصویر بالا است. Parent یک متد همیشه کلاس خواهد بود پس باید کاری شبیه به آنچه در متد اولیه انجام شد را استفاده کنیم و سپس make_parent_class را برای parent متد صدا بزنیم. حالا باید ویژگی های مرتبط به خود متغیر را به دست آوریم:

```

first_parent = ctx.parentCtx
second_parent = first_parent.parentCtx
third_parent = second_parent.parentCtx
idx = 0
for child in first_parent.getChildren():
    if idx == 0:
        self.types.append(child.getText())
        line_col = str(child.start).split(",")[3][:-1].split(':')
        self.lines.append(line_col[0])
        self.columns.append(line_col[1])
        idx += 1
        continue
    if idx == 1:
        var = child.getText().split('=')
        self.names.append(var[0])
        self.values.append(var[1])
        break
has_kind = False
for child in third_parent.getChildren():
    if type(child).__name__ == "ModifierContext":
        self.kind_type.append(child.getText())
        has_kind = True
if not has_kind:
    self.kind_type.append(" ")
self.get_kind_object()

```

برای دستیابی به ویژگی ها به parent اول ، دوم و سوم متغیر نیاز داریم. از parent اول به type, name و value از خود entity متغیر و line و column از refrence خواهیم رسید. از parent سوم نیز به این خواهیم رسید که آیا متغیر ما از kind خاصی است یا نه. در نهایت get_kind_object را صدا می زنیم تا id مرتبط با kind type را به ما بدهد:

```
def get_kind_object(self):
    for kind in self.kind_type:
        if kind == "private":
            self.kind.append(KindModel.get_or_none(_name="Java Variable Private Member")._id)
        elif kind == " ":
            self.kind.append(KindModel.get_or_none(_name="Java Variable Public Member")._id)
```

پس از تمام شدن کار ما در listener لازم است entity ها و refrence را بسازیم:

```
for idx, parent in enumerate(listener.parent_info):
    # make parent entity type of class
    if parent['parent'] == "file":
        file_entity = create_Entity(name, path, None, FileStream(path),
                                   KindModel.get_or_none(_name="Java File")._id, None, None)
        path_parts = path.split('\\')
        i = path_parts.index("src")
        longname = '.'.join(path_parts[i + 1:])
        parent_entity = create_Entity(parent['name'], longname, file_entity, parent['content'],
                                     parent['kind'], None, None)
    # make parent entity type of method
    else:
        parent_class = parent['parent']
        file_entity = create_Entity(name, path, None, FileStream(path),
                                   KindModel.get_or_none(_name="Java File")._id, None, None)
        path_parts = path.split('\\')
        i = path_parts.index("src")
        longname = '.'.join(path_parts[i + 1:])
        class_entity = create_Entity(parent_class['name'], longname, file_entity, parent_class['content'],
                                    parent_class['kind'], None, None)
        longname_method = longname + "." + parent['name']
        parent_entity = create_Entity(parent['name'], longname_method, class_entity, parent['content'],
                                    parent['kind'], None, None)

    var_entity = create_Entity(listener.names[idx], longname + '.' + listener.names[idx],
                              parent_entity, "", listener.kind[idx], listener.values[idx], listener.types[idx])
    create_Ref(KindModel.get_or_none(_name="Java Define")._id, file_entity, listener.lines[idx],
              listener.columns[idx], var_entity, parent_entity)
```

اگر parent از نوع کلاس باشد لازم entity خود parent که class است و entity parent آن کلاس که فایل است را بسازیم. این قسمت if تصویر بالا را تشکیل می دهد. اگر parent از نوع متد باشد لازم است entity متد، parent entity آن یعنی کلاسی که در آن تعریف شده و parent کلاس آن یعنی file را بسازیم. در این مرحله parent ما آماده است و می توانیم entity خود متغیر را با استفاده از همین entity و مواردی که در listener پیدا کردیم بسازیم. حالا همه چیز برای ساخت رفرنس آماده است. Var_entity همان قسمت ent رفرنس را تشکیل می دهد و scope همان parent متغیر خواهد بود. کار ما در این مرحله به اتمام می رسد.

3) modify - modifyby

برای یکی از پروژه های benchmark (پروژه calculator app) به کمک نرم افزار understand اطلاعات مربوط به این رفرنس را در کنسول چاپ میکنیم.

```
E:\anaconda\envs\ understand\python.exe E:/uni/compiler/OpenUnderstand/openunderstand/db/tests.py
Modify i integral.java(27)
ent i ///value 0 ///type int ///ent.kind: Variable ///parent: calculate_pow_x_n ///name: i ///longname: com.calculator.app.method.integral.calculate_pow_x_n.i
scope!! calculate_pow_x_n ///value None ///type double ///ent.kind: Public Method ///parent: integral ///name: calculate_pow_x_n ///longname: com.calculator.app.me
Modify _value_2 integral.java(30)
ent _value_2 ///value 0 ///type double ///ent.kind: Private Variable ///parent: integral ///name: _value_2 ///longname: com.calculator.app.method.integral._value_2
scope!! calculate_pow_x_n ///value None ///type double ///ent.kind: Public Method ///parent: integral ///name: calculate_pow_x_n ///longname: com.calculator.app.me
Modify i integral.java(15)
ent i ///value 0 ///type int ///ent.kind: Variable ///parent: calculate_x ///name: i ///longname: com.calculator.app.method.integral.calculate_x.i
scope!! calculate_x ///value None ///type double ///ent.kind: Public Method ///parent: integral ///name: calculate_x ///longname: com.calculator.app.method.integra
Modify _value_1 integral.java(17)
ent _value_1 ///value 0 ///type double ///ent.kind: Private Variable ///parent: integral ///name: _value_1 ///longname: com.calculator.app.method.integral._value_1
scope!! calculate_x ///value None ///type double ///ent.kind: Public Method ///parent: integral ///name: calculate_x ///longname: com.calculator.app.method.integra
Modify i fibonacci.java(6)
ent i ///value 0 ///type int ///ent.kind: Variable ///parent: calculate_to_n ///name: i ///longname: com.calculator.app.method.fibonacci.calculate_to_n.i
scope!! calculate_to_n ///value None ///type double ///ent.kind: Public Static Method ///parent: fibonacci ///name: calculate_to_n ///longname: com.calculator.app.f
Process finished with exit code 0
```

مطابق تصویر بالا، برای مثال در کد زیر رفرنس modify داریم:

```
public double calculate_pow_x_n(double from, double to, int n) {
    this._value_2 = 0;
    double range = to - from;
    double len = (range / n);
    for (int i = 0; i < n; i++) {
        // pow only is used by this method and this class . pow is a cand
        // method multiplication is being used by two methods of this cla
        this._value_2 += multiplication(len, pow(i, 2));
    }
    new printLog().print("calculated integral is: " + this._value_2);
    return this._value_2;
}
```

منظور از رفرنس modify، متغیرهایی است که در کد دچار تغییر از نوع خاصی شده اند. از جمله این تغییرات به ++، --، +=، -= و ... میتوان اشاره کرد. توکن هایی که شامل این رفرنس میشوند را در یک لیست ذخیره میکنیم.

```
types = [JavaLexer.ADD_ASSIGN, JavaLexer.SUB_ASSIGN, JavaLexer.MUL_ASSIGN, JavaLexer.DIV_ASSIGN,
JavaLexer.AND_ASSIGN, JavaLexer.OR_ASSIGN, JavaLexer.XOR_ASSIGN, JavaLexer.MOD_ASSIGN,
JavaLexer.LSHIFT_ASSIGN, JavaLexer.RSHIFT_ASSIGN, JavaLexer.URSHIFT_ASSIGN, JavaLexer.DEC,
JavaLexer.INC]
```

مطابق تمامی رفرنس های قبل ابتدا تمامی فایل های موجود در دایرکتوری را میخوانیم. و مسیر فایل های جاوا را در یک لیست ذخیره میکنیم. سپس در یک حلقه برای هر یک از فایل ها مراحل زیر را طی میکنیم:

فایل را میخوانیم و برای آن lexer و parser ایجاد میکنیم.

سپس در یک حلقه تمامی توکن های فایل را بررسی میکنیم. در صورتی که از نوعی باشند که باعث modify شدن متغیر هستند، توکن قبلی آن را در یک لیست ذخیره میکنیم. بدین ترتیب نام تمامی متغیرهایی که در فایل مربوطه تغییر میکنند را داریم.

```
previous_token = None
modified_vars = []
for token in tokens.tokens:
    if token.type == JavaLexer.WS:
        continue
    if token.type in types:
        modified_vars.append(previous_token.text)
    previous_token = token
```

اگر طول این لیست برای یک فایل 0 باشد، یعنی در آن فایل رفرنس modify نداریم. پس از آن عبور میکنیم.

```
if len(modified_vars) == 0:
    continue
```

برای این فایل 2 listener میسازیم. اولی مشابه listener به کار رفته در رفرنس define است. بر روی درخت walk میکنیم.

```
tree = parser.compilationUnit()

listener = DefineListener()
scope_listener = ModifyListener()

walker = ParseTreeWalker()
walker.walk(listener=listener, t=tree)
walker.walk(listener=scope_listener, t=tree)
```

اولین listener کارایی دقیقا مشابه define دارد. یعنی تمامی متغیرهای موجود در فایل و اطلاعات مربوط به آن را پیدا میکند. سپس در یک حلقه این متغیرها را بررسی میکنیم. در صورتی که متغیر در لیست modified_vars باشد برای آن مراحل زیر را طی میکنیم:

```

idx = 0
idx2 = 0
for n, parent in zip(listener.names, listener.parent_info):
    if n not in modified_vars:
        idx += 1
        continue

    parent_entity, longname, _ = ent_scope(parent, name, path)

    var_entity = create_Entity(listener.names[idx], longname + '.' + listener.names[idx],
                               parent_entity, "", listener.kind[idx], listener.values[idx],
                               listener.types[idx])

    scope, longname, file_ent = ent_scope(scope_listener.scope_info[idx2], name, path)

    ref = create_Ref(KindModel.get_or_none(name="Java Modify")._id, file_ent, scope_listener.line[idx2],
                    scope_listener.column[idx2], var_entity, scope)

    idx += 1
    idx2 += 1

```

ابتدا با استفاده از متد `ent_scope` برای `entity` مربوطه والد آن را میسازیم (این قسمت مشابه بخشی از کد `define` است). پس از ساختن `parent`، میتوانیم خود `entity` را تشکیل دهیم. سایر اطلاعات مربوط به این `entity` توسط `listener` از پیش پیدا شده است. سپس باید برای ساختن رفرنس `scope` را نیز تشکیل دهیم. برای این کار از یک نمونه از کلاس `ModifyListener` استفاده میکنیم.

```

class ModifyListener(JavaParserLabeledListener):
    def __init__(self):
        self.scopes = []
        self.scope_info = []
        self.line = []
        self.column = []

    def make_scope_method(self, current):
        content = current.getText()
        name = current.IDENTIFIER().getText()
        if type(current).__name__ == "MethodDeclarationContext":
            if "static" in current.getText():
                kind = KindModel.get_or_none(name="Java Static Method Public Member")._id
            else:
                kind = KindModel.get_or_none(name="Java Method Public Member")._id
            current = current.parentCtx
            while current is not None:
                if type(current).__name__ == "ClassDeclarationContext":
                    parent = self.make_scope_class(current)
                    break
                current = current.parentCtx
            self.scope_info.append({"parent": parent, "kind": kind, "name": name, "content": content})

    def make_scope_class(self, current):
        content = current.getText()
        name = current.IDENTIFIER().getText()
        kind = KindModel.get_or_none(name="Java Class Type Public Member")._id
        parent = "file"
        return {"parent": parent, "kind": kind, "name": name, "content": content}

```

```

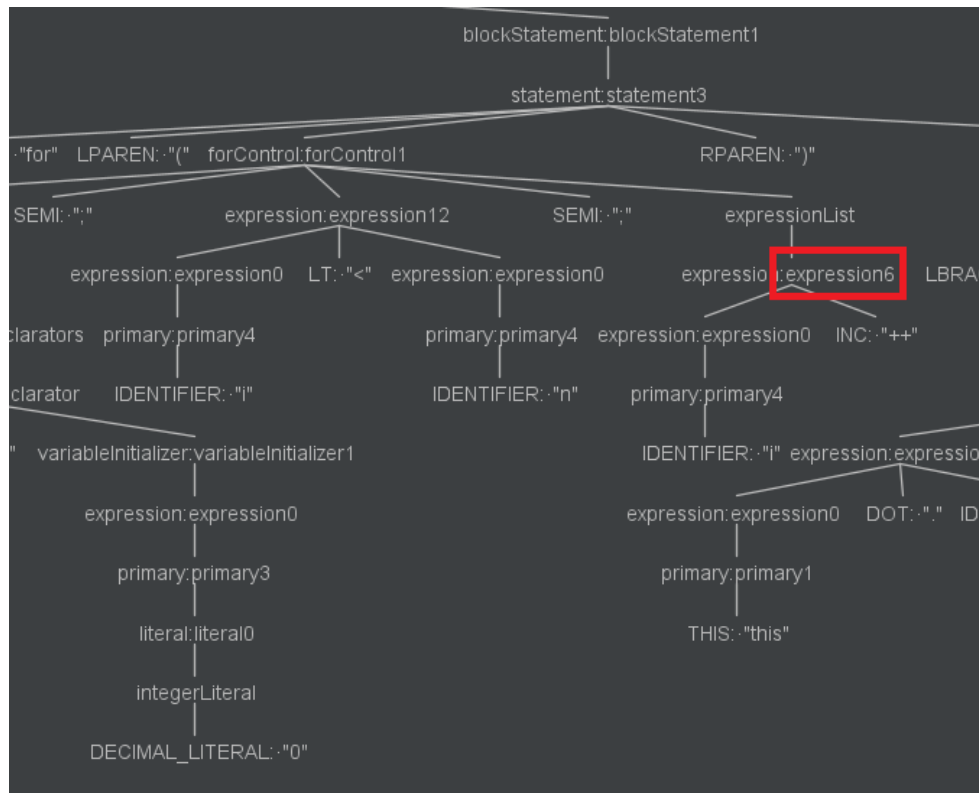
def search_scope(self, ctx):
    current = ctx.parentCtx
    while current is not None:
        if type(current).__name__ == "ClassDeclarationContext" or type(
            current).__name__ == "MethodDeclarationContext":
            self.scopes.append(current.IDENTIFIER().getText())
            if type(current).__name__ == "ClassDeclarationContext":
                self.scope_info.append(self.make_scope_class(current))
            elif type(current).__name__ == "MethodDeclarationContext":
                self.make_scope_method(current)
            return
        current = current.parentCtx
    self.scopes.append(" ")

def enterExpression6(self, ctx: JavaParserLabeled.Expression6Context):
    line_col = str(ctx.children[0].start).split(",")[3][:-1].split(':')
    self.line.append(line_col[0])
    self.column.append(line_col[1])
    self.search_scope(ctx)

def enterExpression21(self, ctx: JavaParserLabeled.Expression21Context):
    operations = ['+=', '-=', '/=', '*=', '&=', '|=', '^=', '%=']
    if ctx.children[1].getText() in operations:
        self.search_scope(ctx)
        line_col = str(ctx.children[0].start).split(",")[3][:-1].split(':')
        self.line.append(line_col[0])
        self.column.append(line_col[1])

```

در اینجا با بررسی درخت مربوط به این رفرنس متوجه میشویم که با استفاده از متدهای `enterExpression6` و `enterExpression21` میتوانیم اسکوپ رفرنس را بیابیم. مثال:



از این دو متد برای یافتن شماره خط و ستونی که در آن رفرنس رخ داده استفاده میکنیم. سایر متدهای موجود در این کلاس، در کلاس DefineListener نیز وجود دارند. از آنها برای یافتن اطلاعات مربوط به اسکوپ و ساختن آن استفاده میکنیم.

```
parent_entity, longname, _ = ent_scope(parent, name, path)

var_entity = create_Entity(listener.names[idx], longname + '.' + listener.names[idx],
                             parent_entity, "", listener.kind[idx], listener.values[idx],
                             listener.types[idx])

scope, longname, file_ent = ent_scope(scope_listener.scope_info[idx2], name, path)

ref = create_Ref(KindModel.get_or_none(name="Java Modify")._id, file_ent, scope_listener.line[idx2],
                 scope_listener.column[idx2], var_entity, scope)
```

سپس از اطلاعات موجود در این کلاس استفاده میکنیم تا scope را بسازیم (خط 159 کد). در پایان نوبت به ساختن reference میرسد. برای این کار از متد create_ref استفاده میکنیم. ورودی های این متد:

ورودی اول kind: برای پیدا کردن kind این رفرنس فایل java_ref_kinds.txt را بررسی میکنیم.

```
63  
64 Modify (Modifyby)  
65  
66 Java Modify  
67  
68
```

ورودی دوم file: به آن فایلی که قبلا entity آن را ساخته بودیم می‌دهیم.
ورودی سوم line: آن را با استفاده از listener قبلا پیدا کردیم.
ورودی چهارم column: آن را با استفاده از listener قبلا پیدا کردیم.
ورودی پنجم entity: به آن entity که قبلا ساختیم می‌دهیم.
ورودی ششم scope: به آن scope که قبلا ساختیم می‌دهیم.