دانشکده مهندسی کامپیوتر

گزارش فاز اول پروژه کامپایلر - گروه ۱۰

اعضای گروه

باوان دیوانی آذر - بیان دیوانی آذر - پارمیدا مجمع صنایع - زهرا مومنی نژاد

اردیبهشت ۱۴۰۱

بسم الله الرحمن الرحيم

فهرست مطالب

مقدمه **4**

نحوه تقسیم کار **5**

روش انجام کار **5**

توضیح بخش‌های مختلف **5**

فایل main.py **5**

فایل declare_declarein_v2 ..... **8**

رفرنس های create, creatby **11**

مقایسه یکی از خروجی understand **و کد ما:** **15**

خروجی understand : **16**

کل خروجی ها : **16**

بررسی یکی از خروجی ها 18

خروجی کد ما: 19

مشکلات و چالش‌ها **21**

مشکلات پروژه **21**

سرعت پایین 21

عدم فرمت بودن کد جاوا در فیلد content دیتابیس 21

نتیجه‌گیری و کارهای آتی **21**

مقدمه

Understand ابزاری قدرتمند برای تجزیه و تحلیل کدها است. در واقع این نرم‌افزار، تمامی موجودیت‌ها که شامل فایل‌ها، پکیج‌ها، توابع، متغیرها و ... است را در پروژه بررسی می‌کند. همچنین نوع رابطه‌های که بین هر کدام از این موجودیتها وجود دارد، خط و نام فایلی که این رابطه در آن رخ داده است و سایر موارد را بررسی می‌کند. ابتدا توضیحات قرار داده شده در سایت را مطالعه کردیم و پروژه را دانلود کردیم و فایل ها را بررسی کردیم همانطور که گفته شده بود understand را نصب کرده و می‌دانیم که این برنامه به ما اجازه می دهد که از API پایتون استفاده کنیم و در واقع ما آدرس های قرار داده را مطابق خواسته ی مسئله تغییر میدهیم و برای این منظور باید حتما این برنامه را نصب کنیم و صرفا نصب داخل برنامه فایده ای ندارد.

مواردی که در پروژه استفاده میشود و بهتر است این جا معرفی شوند:

- Entity: هر چیزی در کد است که Understand اطلاعات مربوط به آن را می گیرد: به عنوان مثال، یک فایل، یک کلاس، یک متغیر، یک تابع و غیره. برای ذخیره موجودیت های جاوا در پروژه استفاده می شود . این جدول در طول تجزیه و تحلیل استاتیک برنامه توسط ANTLR Listener پر می شود. دارای مجموعه ای منحصر به فرد از ویژگی ها است که می تواند توسط API جست و جو شود
- Reference: مکان خاصی که یک موجودیت در کد ظاهر می شود. یک مرجع همیشه به عنوان رابطه بین دو موجودیت تعریف شود.دارای هر دو نهاد مرتبط با آن و همچنین فایل، خط و ستونی است که مرجع در آن قرار می گیرد و نوع مرجعیت آن است. دارای مجموعه ای منحصر به فرد از ویژگی ها است که می تواند توسط API جست و جو شود
- Project: برای ذخیره برخی از اطلاعات اولیه پروژه در حال تحلیل مانند نام پروژه، زبان های برنامه نویسی و غیره. این جدول به صورت خودکار پر می شود.
- Kind: برای ذخیره هر دو نوع موجودیت و مرجع که با پر شدن خودکار دیتابیس و اعداد لازم و مرتبط با بخش های مختلف قابل تفکیک است.

برای بخش پایگاه داده این پروژه از کتابخانه peewee و 3SQLite استفاده شده است و بیشتر داده‌های جمع‌آوری‌شده شامل Entity, Reference است.

نحوه تقسیم کار

ابتدا جلسه‌ای برای بررسی تسک گروه خود تشکیل دادیم. و بعد تقسیم‌بندی زیر انجام شد:

بخش create – createby : باوان دیوانی آذر – بیان دیوانی آذر

بخش declare – declarein : پارمیدا مجمع صنایع – زهرا مومنی نژاد

نوشتن گزارش : همه اعضای گروه

روش انجام کار

ابتدا نرم‌افزار **understand** را نصب کردیم. سپس کد پروژه را از گیت‌هاب کلون کردیم و به مطالعه قسمت‌های مختلف آن پرداختیم. ابتدا فایل **main.py** را بررسی کردیم. که در آن دو بخش استفاده از **listener** هایمان و تابع‌های **addRef** وجود دارد. سپس به پیاده‌سازی **listener** برای هرکدام از تسک‌هایمان پرداختیم. در ادامه گزارش به توضیح جزئی بخش‌های مختلفی که پیاده‌سازی شده می‌پردازیم.

توضیح بخش‌های مختلف

در ادامه گزارش توضیح فایل‌های مختلف به صورت جزئی ذکر می شود.

فایل main.py

در این فایل بخش‌های مختلفی وجود دارد. در ابتدای فایل یک‌سری فایل‌های مورد نظر import شده اند.

```
import os
from fnmatch import fnmatch

from antlr4 import *

from gen.javaLabeled.JavaParserLabeled import JavaParserLabeled
from gen.javaLabeled.JavaLexer import JavaLexer

from oudb.models import KindModel, EntityModel, ReferenceModel
from oudb.api import open as db_open, create_db
from oudb.fill import main

from analysis_passes.couple_coupleby import
ImplementCoupleAndImplementByCoupleBy
from analysis_passes.create_createby import CreateAndCreateBy
from analysis_passes.declare_declarein_v2 import DeclareAndDeclareinListener
from analysis_passes.class_properties import ClassPropertiesListener,
InterfacePropertiesListener
```

سپس در کلاس project به تعریف یک‌سری تابع‌ها پرداخت شده است.

تابع parse که یک فایل را می‌گیرد و token و tree می‌سازد. تابع walk هم که روی درخت پیمایش می‌کند.

تابع getListOfFiles هم تعدادی فایل می‌گیرد و روی آن‌ها یک حلقه for می‌زند و لیستی از مسیر فایل‌ها را برمی‌گرداند.

تابع getFileEntity یک مسیر فایل را می‌گیرد و آن را باز می‌کند. و بعد با تابع get_or_create یک entity به اسم فایل برایمان می‌سازد. و سپس فایل را close می‌کند.

```
class Project():
    tree = None

    def Parse(self, fileAddress):
        file_stream = FileStream(fileAddress)
        lexer = JavaLexer(file_stream)
        tokens = CommonTokenStream(lexer)
        parser = JavaParserLabeled(tokens)
        tree = parser.compilationUnit()
        self.tree = tree
        return tree

    def Walk(self, listener, tree):
        walker = ParseTreeWalker()
        walker.walk(listener=listener, t=tree)

    def getListOfFiles(self, dirName):
        listOfFile = os.listdir(dirName)
        allFiles = list()
        for entry in listOfFile:
            # Create full path
            fullPath = os.path.join(dirName, entry)
            if os.path.isdir(fullPath):
                allFiles = allFiles + self.getListOfFiles(fullPath)
            elif fnmatch(fullPath, "*.java"):
                allFiles.append(fullPath)

        return allFiles

    def getFileEntity(self, path):
        # kind id: 1
        path = path.replace("/", "\\")
        name = path.split("\\")[-1]
        file = open(path, mode='r')
        file_ent = EntityModel.get_or_create(_kind=1, _name=name, _
       longname=path, _contents=file.read())[0]
        file.close()
        print("processing file:", file_ent)
        return file_ent
```

یک تابع دیگر به نام getPackageEntity داریم. که اسم یک file entity و اسم یک پکیج را گرفته و درصورتی که آن موجودیت وجود نداشته باشد آن را می سازد و return می‌کند.

```
def getPackageEntity(self, file_ent, name, longname):
    # package kind id: 72
    ent = EntityModel.get_or_create(_kind=72, _name=name, _parent=file_ent,
                                    _longname=longname, _contents="")
    return ent[0]
```

یک تابع دیگر مشابه تابع بالا وجود دارد که زمانی که اسم یک entity را داریم اما پکیج ما اسم ندارد از آن استفاده می‌شود. که در این حال جای name عبارت unnamed package گذاشته می‌شود. بصورت زیر:

```
def getUnnamedPackageEntity(self, file_ent):
    # unnamed package kind id: 73
    ent = EntityModel.get_or_create(_kind=73, _name="(Unnamed_Package)", _
                                      parent=file_ent,
                                    _longname="(Unnamed_Package)", _
                                      contents="")
    return ent[0]
```

تابع مهم دیگر به نام addDeclareRef است که روابط پیدا شده به دیتابیس اضافه می‌شود.

```
def addDeclareRefs(self, ref_dicts, file_ent):
    for ref_dict in ref_dicts:

        if ref_dict["scope"] is None:  # the scope is the file
            scope = file_ent
        else:  # a normal package
            scope = self.getPackageEntity(file_ent, ref_dict["scope"],
ref_dict["scope_longname"])

        if ref_dict["entity"] is None:  # the entity package is unnamed
            ent = self.getUnnamedPackageEntity(file_ent)
        else:  # a normal package
            ent = self.getPackageEntity(file_ent, ref_dict["entity"],
ref_dict["entity_longname"])

        # Declare reference - kind id = 192
        ReferenceModel.get_or_create(_kind=192, _file=file_ent, _
                                      line=ref_dict["line"],
                                    _column=ref_dict["column"], _ent=ent, _
                                      scope=scope)

        # Declare in reference - kind id = 193
        ReferenceModel.get_or_create(_kind=193, _file=file_ent, _
                                      line=ref_dict["line"],
                                    _column=ref_dict["column"], _scope=ent,
_ent=scope)
```

در آخر هم در تابع main این فایل listener هایی که ایجاد کردیم را صدا می‌زنیم و بعد وقتی declare_dicts ما پر شد تابع addDeclareRef را روی آن صدا می‌زنیم تا به دیتابیس اضافه کنیم.

```
try:
    # declare
    listener = DeclareAndDeclareinListener()
    p.Walk(listener, tree)
    p.addDeclareRefs(listener.get_declare_dicts, file_ent)
    print('declaration dictionary : \n')
    print(listener.get_declare_dicts)
except Exception as e:
    print("An Error occurred for reference declare in file:" + file_address +
"\n" + str(e))
```

فایل declare_declarein_v۲

برای حل مسئله ی مذکور یک فایل با نام declare_declarein_v2 و پسوند py. اینجا کردیم که در آن ابتدا دو بخش مشخص را که مورد استفاده است از پوشه ی gen که ایجاد شده بود برای ما از دو فایل JavaParserLabeledListener و JavaParserLabeled ایمپورت کردیم در ادامه یه کلاس listener تعریف کردیم که در واقع انگار از توابع مورد نیاز یک override کرده و تغییرات و کار های لازم را روی آن ها انجام دادیم

ابتدا کلاسی با اسم DeclareAndDeclareinListener تعریف کردیم و در ادامه برای این کلاس یک constructor نوشتیم که در آن یک متغیری که type آن آرایه است را با نام تعیین کردیم.

بعد دو تابع getter و setter زدیم که در getter صرفا متغیر را برگردانده ایم در setter تابعی تعریف کردیم که برای ما append می‌کند دیتای مورد نظر ما را.

```
class DeclareAndDeclareinListener(JavaParserLabeledListener):

    # define an array for saving entity and reference information
    def __init__(self):
        self.declare_dicts = []

    # getter to return declaration array
    @property
    def get_declare_dicts(self):
        return self.declare_dicts

    # setter to append a data to declaration array
    def set_declare_dicts(self, data):
        self.declare_dicts.append(data)
```

سپس تابع enterCompilationUnit را override کرده و با کمک ctx که در واقع نشان دهنده ی موقعیت و نود ماست کد را تکمیل کردیم.

در اینجا بررسی می‌کنیم که اگر نود ها را پیمایش کردیم و به package نرسیدیم می‌فهمیم که در فایل مورد نظر هیچ پکیجی declare نشده پس ابتدا یک شرط قرار دادیم و در این بخش مشخص کردیم که اگر نود که ما روی آن قرار داریم و در حال بررسی آن هستیم تعیین کننده ی پکیجی نباشد در واقع انگار در این فایل مشخص هیچ پکیجی declare نشده در این صورت باید یک data را به declare_array که این هم در تابع setter قراره داده شده است append کنیم که آن data شامل 4 متغییر است که در آن scope و entity و line و column می‌باشد که این موارد را مطابق شرط که نبودن پکیج است مقدار دهی میکنیم که مقادیر none را داراست در در ردیف و ستون به ترتیب 1 و 0 قرار می‌گیرد.

```
# override enterCompilationUnit function to check if a file declare any
package or not
def enterCompilationUnit(self, ctx:
JavaParserLabeled.CompilationUnitContext):
    if not ctx.packageDeclaration():
        data = {"scope": None, "entity": None, "line": 1, "column": 0}
# unnamed package
        self.set_declare_dicts(data)
```

package_name که یک property از کلاس ماست برابر None قرار می‌دهیم و همچنین این پیام را نمایش می‌دهیم که در این فایل هیچ پکیجی declare نشده است

در ادامه تابع enterPackageDeclaration را override کردیم که فقط در صورتی وارد این تابع می‌شود که نود packageDeclaration مشاهده شود پس کدی که در این تابع زده می‌شود فقط هنگامی اجرا می‌شود که این نود دیده شده و فایل ما پکیجی داشته باشد که در این صورت ما ابتدا نوشتیم که نود مشخص شده چون پکیج داریم یک شاخه دارد به اسم qualifiedName که اسم پکیج را به ما می‌دهد که ما از تابع getText استفاده کردیم که در خروجی به صورت string اسم پکیج را برگرداند و این اسم را در متغییری به نام full_package_name_array ذخیره کردیم.

حال متغییر با نام longname انتخاب کرده و برابر string خالی می‌گذاریم، میخواهیم یک for بزنیم و چون در طول متغیر برگردانده شده می‌خواهیم حرکت کنیم پس به طول full_package_name_array این for را میزنیم و می‌دانیم که مواردی ذکر شده در فوق را یکی یکی به ما برمی‌گرداند که ما می‌خواهیم مرتب یه خواسته

ی خودمان با دات گذاری بنویسیم پس ابتدا می‌خواهیم که هر عضو آرایه ی ما را که یک کلمه هست برای ما getText کند و یک متغیر جدید با نام entity_longname مشخص می‌کنیم که برای ما بعد longname اگر خالی نباشد دات می‌گذارد و اگر خالی باشد string خالی قرار می‌دهد و در نهایت با entity_name جمع می‌کند.

در قسمت فوق که هیچ پکیجی declare نشده بود ما تمامی متغیر ها را جداگانه مقدار دهی کردیم در این بخش نیز مجددا این کار را میکنیم که در کد مشخص شده است که هر متغیر به چه صورت مقداردهی شده است و در خط ابتدایی که scope را مقدار دهی کردیم از i -1 استفاده کردیم زیرا یک مقدار کمتر از چیزی است که ما میخوایم پس اشاره می‌کنیم به قبلی آن و به همین شکل ادامه داده و متغیر های دیگر را اسم گذاری کردیم.

در نهایت این data را با مقداردهی های مشخص به آن آرایه ی ابتدایی توسط تابع setter اضافه کردیم و خط آخر نیز متغیر entity_longname را داخل longname ریختیم و پاسخ مسئله ی خواسته شده مطابق توضیحات فوق تکمیل شد.

در کد نیز همانطور که عکس های آن گذاشته شده است در هر بخش متناسب با فعالیت انجام شده کامنت‌ها قرار داده شده اند. حال هر دو بخش خواسته شده ی مسئله را بررسی کردیم و اگر فایلی به ما داده شود می‌توانیم بررسی کنیم که آیا پکیجی در آن declare شده یا نه.

```
# override enterPackageDeclaration function to set reference
information
def enterPackageDeclaration(self, ctx:
JavaParserLabeled.PackageDeclarationContext):
    full_package_name_array = ctx.qualifiedName().IDENTIFIER()

    longname = ""
    for i in range(len(full_package_name_array)):
        entity_name = full_package_name_array[i].getText()
        entity_longname = longname + ("." if longname != "" else "") +
entity_name
        [line, column] = str(ctx.start).split(",")[3].split(":")
        data = {
            "scope": full_package_name_array[i - 1].getText() if i != 0
else None, "entity": entity_name,
            "scope_longname": longname, "entity_longname":
entity_longname,
            "line": line, "column": column.strip("]")
        }
        self.set_declare_dicts(data)
        longname = entity_longname
```

رفرنس های **create, creatby**

ابتدا یک Listener مینویسیم که هنگامی در یک تابع یک شی از یه کلاس ساخته میشود رفرنس های create , createby را بررسی کند.

برای مثال:

```
class c1 {
 ...
}

class c2 {
 c1 a = new c1();
}
```

Reference kind string	Entity performing references	Entity being referenced
Java Create	c2	c1
Java Createby	c1	c2

شکل 1

در اینجا شی c1 در کلاس c2 ساخته شده و رفرنس create by توسط c1 به c2 و رفرنس create توسط c2 به c1 داریم.

مراحل پیاده سازی Listener :

- کلاس Listener خود را از کلاس Listener از پیش پیاده سازی شده انتلر یعنیJavaParserLabeledListener ارث‌بری میکنیم.
- آرایه create برای ذخیره‌سازی اطلاعات لازم میسازیم.
- حال به کمک understand بررسی میکنیم که وقتی create اتفاق می‌افتد چه اطلاعاتی به ما میدهد. فایلtests.py را ادیت میکنیم و رفرنس های create را بررسی میکنیم، متوجه میشویم که برای create به modifier که نمایانگر public یا private بودن entity هست احتیاج داریم که با کمک تابع get_method_modifier آن را بدست میاوردیم.

- در این تابع ابتدا از فرزند به سمت parent ها پیمایش میکنیم که به rule از نوع class body Declaration برسیم و سپس به کمک آن modifier هارا بدست می آوردیم

```
def get_method_modifiers(self, c):
    parents = ""
    modifiers = []
    current = c
    while current is not None:
        if "ClassBodyDeclaration" in type(current.parentCtx).__name__:
            parents = (current.parentCtx.modifier())
            break
        current = current.parentCtx
    for x in parents:
        if x.classOrInterfaceModifier():
            modifiers.append(x.classOrInterfaceModifier().getText())
    return modifiers
```

کد 1-1-1

با استفاده از understand فهمیدیم مقدار content هر entity برابر است با کد تابعی که آن entity در آنجا قرار دارد. چون نتایج را db browser کامل نشان نمیدهد ما پرینت شان کردیم.

```
def get_method_content(self, c):
    parents = ""
    content = ""
    current = c
    while current is not None:
        if type(current.parentCtx).__name__ ==
"ClassBodyDeclaration2Context":
            content = current.parentCtx.getText()
            break
            parents = (current.parentCtx.typeTypeOrVoid().getText())
        current = current.parentCtx

    print(f"Entity contex : {content}")

    return parents, content
```

کد 1-1-2

برای گرفتن اسم پکیج (برای ساختن longname احتیاج داریم) تابع زیر را override می‌کنیم.

```
def enterPackageDeclaration(self, ctx:
JavaParserLabeled.PackageDeclarationContext):
    self.package_name = ctx.qualifiedName().getText()
```

کد 3-1-1

برای گرفتن اسم کلاس تابع زیر را override می‌کنیم.

```
def enterClassDeclaration(self, ctx:
JavaParserLabeled.ClassDeclarationContext):
    self.class_name = ctx.IDENTIFIER().getText()
```

کد 4-1-1

با چک کردن parse tree فهمیدیم هروقت یک child از کلاس new می‌شود

در شاخه Expression4Context رخ می‌دهد.

پس اسم کلاسی که ازش child ساختیم در خط دوم بدست می آوریم در new_class_name ذخیره می‌کنیم.

line , col از طریق ctx.start بدست می آوریم.

```
def enterExpression4(self, ctx: JavaParserLabeled.Expression4Context):

    new_class_name = ctx.creator().createdName().IDENTIFIER()[0].getText()

    if not self.refers.__contains__(self.class_name):
        self.refers[self.class_name] = []
    self.refers[self.class_name].append(new_class_name)

    if ctx.creator().classCreatorRest():
        all_refs = class_properties.ClassPropertiesListener.findParents(ctx)
        [line, col] = str(ctx.start).split(",")[3].split(":")

        modifiers = self.get_method_modifiers(ctx)
        method_return, method_context = self.get_method_content(ctx)

        self.create.append(
            {"scope_name": all_refs[-1], "scope_longname":
".".join(all_refs), "scope_modifiers": modifiers,
             "scope_return_type": method_return, "scope_content":
method_context,
             "line": line, "col": col[:-1], "refent": new_class_name,
             "scope_parent": all_refs[-2] if len(all_refs) > 2 else None,
             "potential_refent": ".".join(
                 all_refs[:-1]) + "." + new_class_name, "package_name":
self.package_name})
```

کد 5-1-1

در main.py در تابع main بین فایل های پروژه iteration می‌کنیم. و هردفعه هر فایلی که بررسی می‌کنیم به عنوان entity به جدول entitymodel دیتابیس اضافه می‌کنیم

اینجا listener ای که ساختم به listener می‌دهیم بعد tree را walk می‌کنیم. و تابع addCreateRefs صدا میزنیم و به عنوان ورودی دیکشنریcreate و فایل و آدرس آن را می‌دهیم.

```
if __name__ == '__main__':
    p = Project()
    create_db("../benchmark2_database.oudb",
              project_dir="..\benchmark")
    main()
    db = db_open("../benchmark2_database.oudb")
    # get file name
    rawPath = str(os.path.dirname(__file__).replace("\\", "/"))
    pathArray = rawPath.split('/')
    path = Project.listToString(pathArray) + "benchmark"
    files = p.getListOfFiles(path)

    for file_address in files:
        try:
            file_ent = p.getFileEntity(file_address)
            tree = p.Parse(file_address)
        except Exception as e:
            print("An Error occurred in file:" + file_address + "\n" +
str(e))
            continue
        try:
            listener = CreateAndCreateByListener()
            p.Walk(listener, tree)
            listener.get_refers()
            p.addCreateRefs(listener.get_create(), file_ent, file_address)
        except Exception as e:
            print("An Error occurred for reference implement in file:" +
file_address + "\n" + str(e))
```

کد 6-1-1

تابع addCreateRefs

```
def addCreateRefs(self, ref_dicts, file_ent, file_address):
    for ref_dict in ref_dicts:
        scope =
EntityModel.get_or_create(_kind=self.findKindWithKeywords("Method",
ref_dict["scope_modifiers"]),
                                                _name=ref_dict["scope_name"],

_type=ref_dict["scope_return_type"]
                                                ,
_parent=ref_dict["scope_parent"] if ref_dict["scope_parent"] is not
None else file_ent
                                                ,
_longname=ref_dict["package_name"]+"."+ref_dict["scope_longname"]
                                                ,
_contents=["scope_content"])[0]
        ent =
self.getCreatedClassEntity(ref_dict["package_name"]+"."+ref_dict["refen
t"], ref_dict["potential_refent"], file_address)
        Create = ReferenceModel.get_or_create(_kind=190,
_file=file_ent, _line=ref_dict["line"],
                                                    _column=ref_dict["col"],
_scope=scope, _ent=ent)
        Createby = ReferenceModel.get_or_create(_kind=191,
_file=file_ent, _line=ref_dict["line"],

_column=ref_dict["col"], _scope=ent, _ent=scope)
```

کد 7-1-1

مقایسه یکی از خروجی **understand** و کد ما:

اگر خروجی 3-1-2 و 2-2-2 را مقایسه کنیم longname هر دو مثل هم هستند ما تو دیتابیس برای kind به جدول kindmodel رفرنس دادیم. ما public method تو جدول kindmodel نداشتیم و جایگزین آن طبق خروجی 4-2-2 Java Method Public member داشتیم برای Public Class هم طبق خروجی 5-2-2 Java Unknown Class Type member هست..

با توجه به خروجی 2-2-2 ما شماره خط هم درست بدست آوردیم.

درنتیجه kind هم مانند خروجی Understand هست

فایل هم به جدول entitymodel رفرنس دادیم (چون که فایل هم یک entity حساب می‌شود.)

با مقایسه خروجی های 4-1-2 و 6-2-2 به مشابه بودن context هم پی می‌بریم (البته ما کامنت حساب نکردیم)

خروجی understand :

در فایل tests.py ما مقدار هایی که می‌خواستیم در und پرینت کردیم

```
def test_understand_kinds():
  db=nd.open(r"C:\Users\ASUS\Desktop\term_4002\Compiler\project\phase1\New folder\OpenUnderstand\benchmark\calculator_app\calculator_app2.und")
    for ent in db.ents():
        for ref in ent.refs("Create"):
            print(f'ref.scope (entity performing reference)\t: "{ref.scope().longname()}", kind: "{ref.scope().kind()}"')
            print(f'ref.ent (entity being referenced)\t\t: "{ref.ent().longname()}", kind: "{ref.ent().kind()}"')
            print(f'File where the reference occurred: "{ref.file().longname()}", line: {ref.line()}')
```

کد 1-1-2

کل خروجی ها :

```
ref.scope (entity performing reference)	: "com.calculator.app.method.integral.calculate_pow_x_n", kind: "Public Method"

ref.ent (entity being referenced)		: "com.calculator.app.method.printLog", kind: "Public Class"

File where the reference occurred: "C:\Users\ASUS\Desktop\term_4002\Compiler\project\phase1\New folder\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\integral.java", line: 32

-------------------------------------------------

ref.scope (entity performing reference)	: "com.calculator.app.method.integral.calculate_x", kind: "Public Method"

ref.ent (entity being referenced)		: "com.calculator.app.method.printLog", kind: "Public Class"
```

File where the reference occurred: "C:\Users\ASUS\Desktop\term_4002\Compiler\project\phase1\New folder\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\integral.java", line: 19

---

ref.scope (entity performing reference) : "com.calculator.app.display.println.print_fail", kind: "Public Method"

ref.ent (entity being referenced) : "com.calculator.app.display.print_fail", kind: "Public Class"

File where the reference occurred: "C:\Users\ASUS\Desktop\term_4002\Compiler\project\phase1\New folder\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\println.java", line: 27

---

ref.scope (entity performing reference) : "com.calculator.app.init.Main.main", kind: "Public Static Method"

ref.ent (entity being referenced) : "com.calculator.app.method.integral", kind: "Public Class"

File where the reference occurred: "C:\Users\ASUS\Desktop\term_4002\Compiler\project\phase1\New folder\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\init\Main.java", line: 12

---

ref.scope (entity performing reference) : "com.calculator.app.init.Main.main", kind: "Public Static Method"

ref.ent (entity being referenced) : "com.calculator.app.display.println", kind: "Public Class"

File where the reference occurred: "C:\Users\ASUS\Desktop\term_4002\Compiler\project\phase1\New folder\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\init\Main.java", line: 17

خروجی 2-1-2

بررسی یکی از خروجی ها

ref.scope (entity performing reference) : "com.calculator.app.display.println.print_fail", kind: "Public Method"

ref.ent (entity being referenced) : "com.calculator.app.display.print_fail", kind: "Public Class"

File where the reference occurred: "C:\Users\ASUS\Desktop\term_4002\Compiler\project\phase1\New folder\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\println.java", line: 27

Entity longname: com.calculator.app.display.println.print_fail

Entity parent: println

Entity kind: Public Method

Entity value: None

Entity type: void

خروجی 2-1-3

Entity contents:

```
public void print_fail() {
    // we intentionally made print_fail_message non-static
    print_fail pf = new print_fail();
    pf.print_fail_message();
}
```

خروجی 2-1-4

خروجی کد ما:

Entitymodel :

	_id	_kind_id	_parent_id	_name	_longname	_value	_type	_contents
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	NULL	println.java	C:...	NULL	NULL	package ...
2	2	39	1	print_fail	com.calculator.app.display.println.print_fail	NULL	void	['scopecontent']
3	3	84	NULL	print_fail	com.calculator.app.display.print_fail	NULL	NULL	

خروجی 1-2-2

Referencemodel:

Table: referencemodel Filter in any column

	_id	_kind_id	_file_id	_line	_column	_ent_id	_scope_id
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	190	1	27	24	3	2
2	2	191	1	27	24	2	3

خروجی 2-2-2

Kindmodel:

Table: kindmodel Filter in any

	_id	_inv_id	_name	is_ent_kind
	Filter	Filter	Filter	Filter
187	187	186	Java Extend Coupleby Implicit	0
188	188	NULL	Java Implement Couple	0
189	189	188	Java Implement Coupleby	0
190	190	NULL	Java Create	0
191	191	190	Java Createby	0

خروجی 3-2-2

38	38	NULL	Java Unresolved External Final Method ...	1
39	39	NULL	Java Method Public Member	1

خروجی 4-2-2

84	84	NULL	Java Unknown Class Type Member	1
85	85	NULL	Java Abstract Generic Class Type Public	1

خروجی 5-2-2

processing file: println.java

Entity contex :
publicvoidprint_fail(){print_failpf=newprint_fail();pf.print_fail_message();}

خروجی 6-2-2

مشکلات و چالش‌ها

چالش اصلی که برخوردیم دیدن نتایج understand بود که با نسخه کرکی که در بیشتر سایتا بود ۶.۱ هر چقدر تلاش کردیم نتونستیم و به مشکل لایسنس میخوردیم بعد به کمک یکی از بچه ها فهمیدیم که میتوانیم مستقیم از سایت scitools لایسنس دانش‌آموزی بگیریم و به کمک آن مشکلمان رفع شد.

مشکل دوم تسک دوم گروه declare declare in بود که تو داکیومنت TODO بود و ما بعد از پیگیری در داک قرار داده شد اما متاسفانه ما یک برداشت غلطی ازش کردیم و کدی که زدیم خیلی درست نبود و این برای زمانی بود که ما نمیدونستیم میتوانیم با understand نتایج درست را مشاهده کنیم با راهنمایی گرفتن از منتور گروه خانم منصوری کدمان را اصلاح کردیم و این مشکل هم حل شد.

مشکلات پروژه

سرعت پایین

در پروژه تعداد زیادی Listener استفاده شده به همین خاطر سرعت پروژه کم شده.

عدم فرمت بودن کد جاوا در فیلد content دیتابیس

متاسفانه کدی که به ما api understand میدهد whitespace ها و newline ها حذف شده بنابراین کد به زیبایی کدی که در نتایج نرم‌افزار understand میتوانیم مشاهده کنیم نیست.

نتیجه‌گیری و کارهای آتی

به کمک موارد پیاده سازی شده میتوانیم روابط create و declare را در کد های جاوا به خوبی تجزیه و تحلیل کرد.