



امین حسن زارعی

شماره دانشجویی

۴۰۱۷۲۳۰۹۴

استاد راهنما

دکتر سعید پارسا

آبان ۱۴۰۲

فهرست مطالب

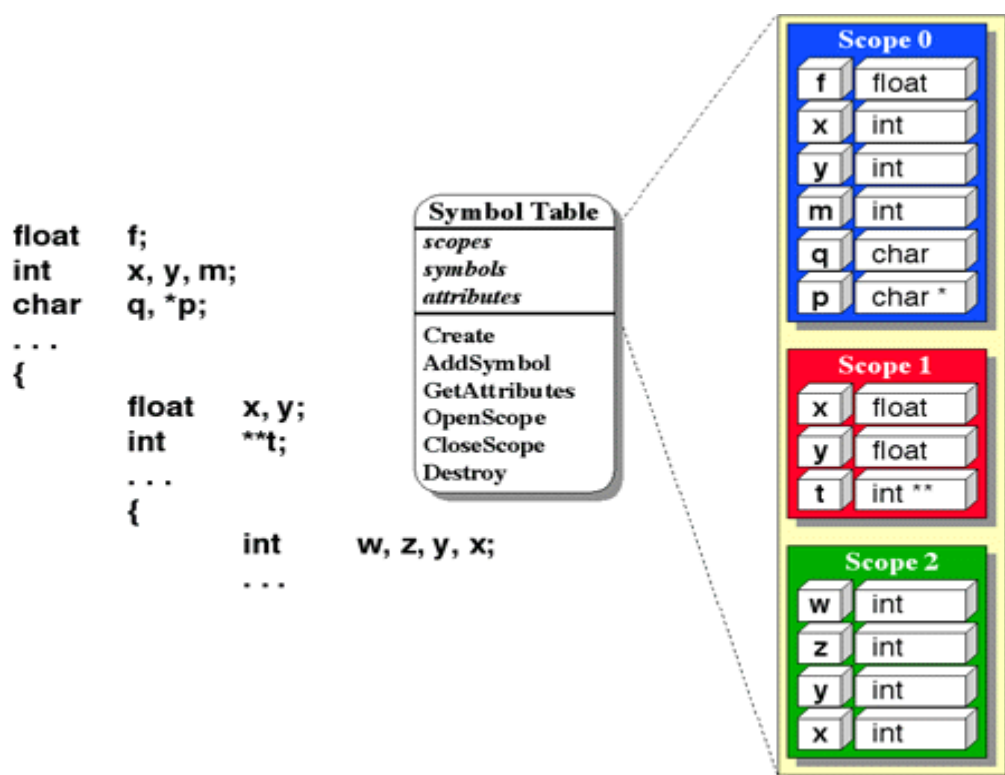
۴	۱. جدول نمادها
۵	۱-۱. مثالی از ساخت یک جدول نمادها و قرار دادن آنها در پایگاه داده:
۷	۲. معیارهای نرم افزاری
۸	۲-۱. مثالی از محاسبه معیار نرم افزاری Line of code
۱۰	۳. نرم افزار understand چیست؟
۱۰	۳-۱. مثالی از استفاده از api call نرم افزار understand در زبان python
۱۱	۳-۲. مثالی از ساخت جدول نمادها با استفاده از understand
۱۱	۳-۳. مثالی از محاسبه متریک ها در نرم افزار understand
۱۲	۴. پیشنهاد های توسعه نرم افزار openunderstand
۱۳	۵. معماری نرم افزار openunderstand
۱۳	۵-۱. موجودیت و مرجع
۱۵	۵-۱-۱. انواع موجودیت
۱۶	۵-۱-۲. انواع مرجع
۱۶	۵-۲. طرحواره پایگاه داده ERD
۱۸	۶. یک مثال کلی که نرم افزار openunderstand چگونه کار می کند
۱۹	۷. نحوه راه اندازی نرم افزار openunderstand
۲۶	۸. اضافه کردن type به جدول نمادها

۸-۱ ذخیره کردن مدل ها ۲۷

۹. اضافه کردن api معیار ۳۰

۱۰. لینک های مفید ۳۵

۱. جدول نمادها



جدول نمادها^۱ ساختار داده ای است که توسط کامپایلرها، مفسرها و سایر سیستم های پردازش زبان برای مدیریت اطلاعات مربوط به شناسه های برنامه، مانند متغیرها، توابع، کلاس ها و برچسب ها استفاده می شود. این به عنوان یک جدول جستجو عمل می کند که هر شناسه را با ویژگی های مربوطه مرتبط می کند و بازایی و مدیریت کارآمد اطلاعات مربوط به نماد را در طول مراحل مختلف تدوین یا اجرای برنامه تسهیل می کند.

جدول نمادها معمولاً اطلاعاتی مانند نام شناسه، نوع داده، مکان یا آدرس حافظه، دامنه یا قابلیت مشاهده و سایر ویژگی های مرتبط را ذخیره می کند. برای کارهایی مانند وضوح نام، بررسی نوع و تولید کد استفاده می شود.

^۱ Symbol table

جدول نمادها مانند یک سیستم بایگانی سازماندهی شده است، واکنشی سریع اطلاعات مربوط به شناسه ها را در صورت نیاز برای کامپایلر یا مترجم آسان تر می کند.

۱-۱. مثالی از ساخت یک جدول نمادها و قرار دادن آنها در پایگاه داده:

```
import sqlite3
from antlr4 import *
from JavaLexer import JavaLexer
from JavaParser import JavaParser
from JavaParserListener import JavaParserListener

# Symbol table class
class SymbolTable(JavaParserListener):
    def __init__(self):
        self.symbols = []

    def enterVariableDeclaratorId(self, ctx:
        JavaParser.VariableDeclaratorIdContext):
        symbol_name = ctx.getText()
        self.symbols.append(symbol_name)

    def enterMethodDeclaration(self, ctx:
        JavaParser.MethodDeclarationContext):
        method_name = ctx.IDENTIFIER().getText()
        self.symbols.append(method_name)

# SQLite helper class
class SQLiteHelper:
    def __init__(self, db_file):
        self.db_file = db_file
        self.conn = None
        self.cursor = None

    def connect(self):
        self.conn = sqlite3.connect(self.db_file)
        self.cursor = self.conn.cursor()

    def disconnect(self):
        if self.conn:
            self.conn.close()
```

```

def create_table(self):
    self.cursor.execute("CREATE TABLE IF NOT EXISTS symbols
(name TEXT)")

def insert_symbol(self, symbol):
    self.cursor.execute("INSERT INTO symbols (name) VALUES
(?)", [symbol])
    self.conn.commit()

# Main program
def main():
    # Create an empty symbol table
    symbol_table = SymbolTable()

    # Create a parser for the Java application
    input_stream = FileStream('path/to/your/java/file.java')
    lexer = JavaLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = JavaParser(stream)

    # Traverse the parse tree and populate the symbol table
    walker = ParseTreeWalker()
    walker.walk(symbol_table, parser.compilationUnit)

    # Store the symbols in an SQLite database
    db_file = 'symbol_table.db'
    db_helper = SQLiteHelper(db_file)
    db_helper.connect()
    db_helper.create_table()

    for symbol in symbol_table.symbols:
        db_helper.insert_symbol(symbol)

    db_helper.disconnect()

    print("Symbol table created and stored in SQLite database.")

if __name__ == '__main__':
    main()

```

۲. معیارهای نرم افزاری



معیارهای نرم افزاری^۲ معیارهای کمی هستند که برای ارزیابی ویژگی های مختلف سیستم های نرم افزاری و فرآیند توسعه نرم افزار استفاده می شوند. آنها اطلاعات عینی در مورد کیفیت، اندازه، پیچیدگی، کارایی، قابلیت نگهداری و سایر ویژگی های نرم افزار ارائه می دهند. معیارهای نرم افزار برای تصمیم گیری آگاهانه، نظارت بر پیشرفت، شناسایی مسائل بالقوه و بهبود شیوه های توسعه نرم افزار استفاده می شود.

معیارهای نرم افزاری انواع مختلفی دارند، از جمله:

معیارهای اندازه: این معیارها اندازه نرم افزار را معمولاً در خطوط کد (LOC)، تعداد نقاط تابع یا سایر واحدهای اندازه گیری اندازه گیری می کنند.

معیارهای پیچیدگی: این معیارها پیچیدگی سیستم های نرم افزاری را ارزیابی می کنند، مانند تعداد مسیرهای کنترل جریان، پیچیدگی چرخه ای یا سطح تودرتو کد.

^۲ Software metrics

معیارهای کیفیت: معیارهای کیفیت بر ویژگی های کیفیت نرم افزار مانند قابلیت اطمینان، قابلیت نگهداری، قابلیت استفاده، عملکرد و امنیت تمرکز می کنند. به عنوان مثال می توان به تراکم نقص، پوشش کد و رتبه بندی رضایت مشتری اشاره کرد.

معیارهای تلاش: معیارهای تلاش، زمان، منابع و تلاش مورد نیاز برای فعالیت های توسعه نرم افزار را کمیت می کنند. آنها می توانند شامل معیارهایی مانند ساعت کار، زمان توسعه یا هزینه باشند.

معیارهای بهره وری: معیارهای بهره وری بهره وری و کارایی افراد یا تیم های درگیر در توسعه نرم افزار را ارزیابی می کنند. آنها ممکن است عواملی مانند خطوط کد تولید شده در ساعت یا تعداد داستان های تکمیل شده کاربر را در نظر بگیرند.

با جمع آوری و تجزیه و تحلیل معیارهای نرم افزار، سازمان ها می توانند بینشی در مورد فرآیند توسعه نرم افزار به دست آورند، زمینه های بهبود را شناسایی کنند، پیشرفت را در طول زمان ردیابی کنند و تصمیمات مبتنی بر داده ها را برای افزایش کیفیت نرم افزار و کارایی توسعه اتخاذ کنند.

معیارهای نرم افزار به سازمان ها کمک می کند تا تلاش های توسعه نرم افزار خود را اندازه گیری، درک و بهبود بخشند.

۲-۱. مثالی از محاسبه معیار نرم افزاری Line of code

```
from antlr4 import *
from JavaLexer import JavaLexer
from JavaParser import JavaParser

# Metric Calculator class
class MetricCalculator(JavaParserListener):
    def __init__(self):
        self.loc = 0

    def enterClassDeclaration(self, ctx:
        JavaParser.ClassDeclarationContext):
        class_lines = ctx.stop.line - ctx.start.line + 1
        self.loc += class_lines

    def enterMethodDeclaration(self, ctx:
        JavaParser.MethodDeclarationContext):
        method_lines = ctx.stop.line - ctx.start.line + 1
        self.loc += method_lines
```



```

def enterVariableDeclarator(self, ctx:
JavaParser.VariableDeclaratorContext):
    variable_lines = ctx.stop.line - ctx.start.line + 1
    self.loc += variable_lines

def get_loc(self):
    return self.loc

# Main program
def main():
    # Create a metric calculator
    metric_calculator = MetricCalculator()

    # Create a parser for the Java application
    input_stream = FileStream('path/to/your/java/file.java')
    lexer = JavaLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = JavaParser(stream)

    # Traverse the parse tree and calculate the LOC metric
    walker = ParseTreeWalker()
    walker.walk(metric_calculator, parser.compilationUnit)

    loc = metric_calculator.get_loc()

    print("Lines of Code (LOC) metric:", loc)

if __name__ == '__main__':
    main()

```

۳. نرم افزار understand چیست؟



SciTools شرکتی است که ابزارها و راه حل های توسعه نرم افزار را برای برنامه های کاربردی علمی و مهندسی ارائه می دهد. آنها طیف وسیعی از محصولات را با تمرکز بر تجزیه و تحلیل کد، آزمایش و تجسم ارائه می دهند تا به توسعه دهندگان در بهبود کیفیت و عملکرد کدشان کمک کنند.

یکی از محصولات محبوب آنها Understand است که یک ابزار درک کد است Understand. به توسعه دهندگان اجازه می دهد تا پایگاه های کد نوشته شده در زبان های برنامه نویسی مختلف مانند C++، C#، Python، Java و غیره را تجزیه و تحلیل کنند. با تولید معیارها، نمودارهای وابستگی و سایر تجسمها، درک عمیقی از کد ارائه می کند.

به کمک Understand توسعه دهندگان می توانند بینش هایی درباره ساختار، پیچیدگی و روابط درون پایگاه کد کسب کنند. این به شناسایی مشکلات احتمالی، مانند کد تکراری، کد مرده، یا مناطق کد پیچیده که نیاز به refactoring دارند، کمک می کند. همچنین ردیابی تغییرات کد، انجام تجزیه و تحلیل تاثیر، و ایجاد گزارش در معیارهای کیفیت کد را امکان پذیر می کند.

به طور کلی، Understand 'SciTools' ابزار قدرتمندی است که به توسعه دهندگان در درک پایگاه های کد و تصمیم گیری آگاهانه مربوط به کیفیت، نگهداری و بهینه سازی کد کمک می کند.

۳-۱. مثالی از استفاده از api call نرم افزار understand در زبان python:

```
import understand

# Open the codebase using Understand
db = understand.open("path/to/codebase.udb")
```

```
# Get a list of all files in the codebase
files = db.ents("file")

# Iterate over each file and print its name
for file in files:
    print(file.name())

# Close the codebase
db.close()
```

۳-۲. مثالی از ساخت جدول نماد ها با استفاده از understand

```
import subprocess

# Replace "<path_to_understand>" with the actual path to the
'und' command line interface
understand_cmd = "<path_to_understand>/und"

# Specify the command-line arguments to create a UDB file for
Java
create_udb_args = [
    understand_cmd,
    "create",
    "-languages",
    "Java",
    "<path_to_source_files>",
    "<path_to_output_udb>"
]

# Run the command to create the UDB file
subprocess.run(create_udb_args)
```

۳-۳. مثالی از محاسبه متریک ها در نرم افزار understand

```
import understand
for class_entity in db.ents('class'):
    loc = class_entity.metric(['CountLineCode']) # Retrieve the
LOC metric
```

```
print(f"{class_entity.longname()}: {loc}")
db_path = '/path/to/your/project.udb' # Specify the path to
your Understand database file
db = understand.open(db_path)
db.close()
```

۴. پیشنیاز های توسعه نرم افزار openunderstand



شکل ۴.۱ Git



شکل ۴.۲ Python



شکل ۴.۴ SQLite



شکل ۴.۳ Antlr

۵. معماری نرم افزار openunderstand

۵-۱. موجودیت و مرجع

بیشتر داده های جمع آوری شده توسط Understand شامل Entities و References است.

Entity: Entity هر موجودیت در کد است که Understand اطلاعات مربوط به آن را می گیرد: به عنوان مثال، یک فایل، یک کلاس، یک متغیر، یک تابع و...، در Perl API، موجودیت ها با کلاس Understand::Ent نشان داده می شوند. در پایتون، کلاس Understand.Ent است.

Reference: مکان خاصی که یک موجودیت در کد ظاهر می شود. مرجع همیشه به عنوان رابطه بین دو موجودیت را تعریف می کند. به عنوان مثال، نوار تابع در خط ۱۴ تابع Foo فراخوانی می شود. در Perl API، مراجع با کلاس Understand::Ref نشان داده می شوند. در پایتون، کلاس Understand.Ref است.

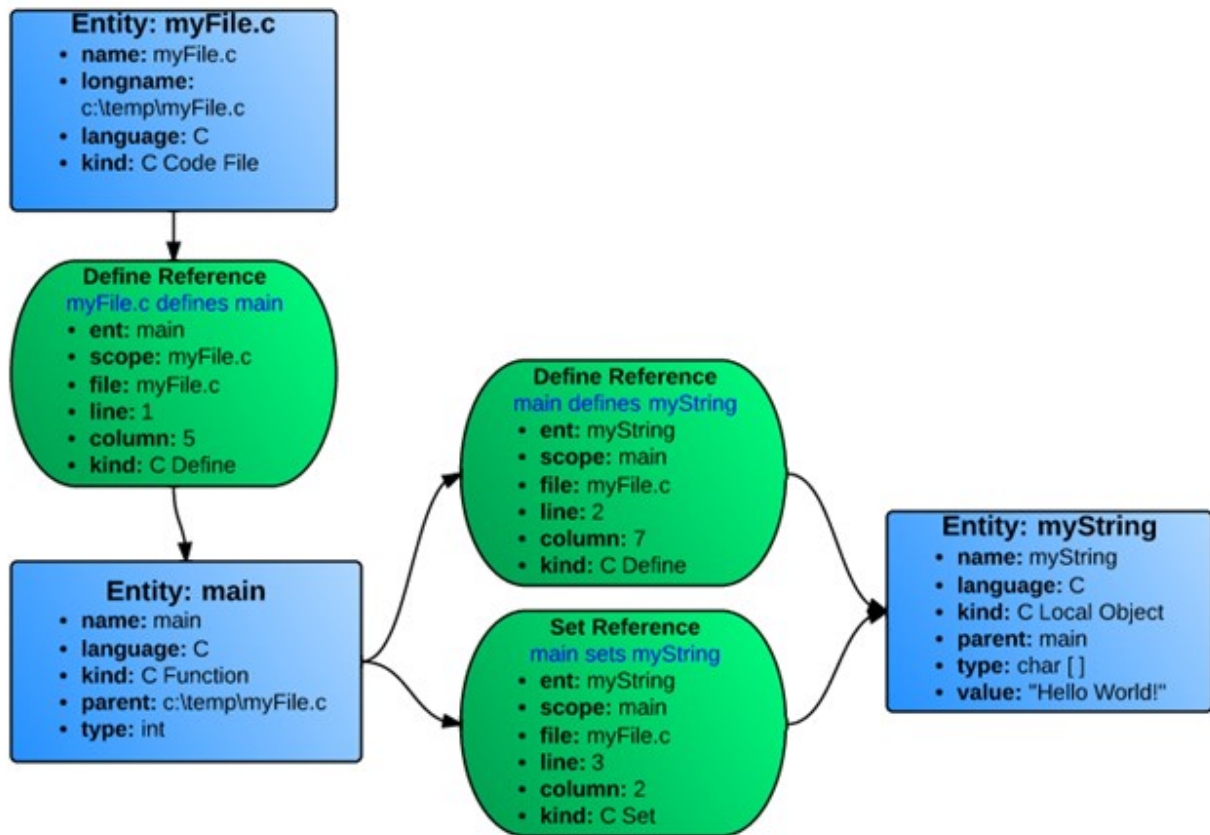
هر موجودیت و مرجع دارای مجموعه ای منحصر به فرد از ویژگی ها است که می تواند توسط API جستجو شود. تعدادی از ویژگی هایی که می توانید برای یک موجودیت مشاهده کنید عبارتند از: نام، نوع آن، هر مرجع مرتبط، نوع موجودیت آن، و اینکه آیا آنها را دارد: موجودیت اصلی و پارامترهای آن. از سوی دیگر، یک مرجع دارای هر دو موجودیت مرتبط با آن و همچنین فایل، خط و ستونی است که مرجع در آن قرار می گیرد و نوع مرجع آن چیست.

برای کمک به تجسم این موضوع، بیاید از این کد C ساده استفاده کنیم:

myFile.c

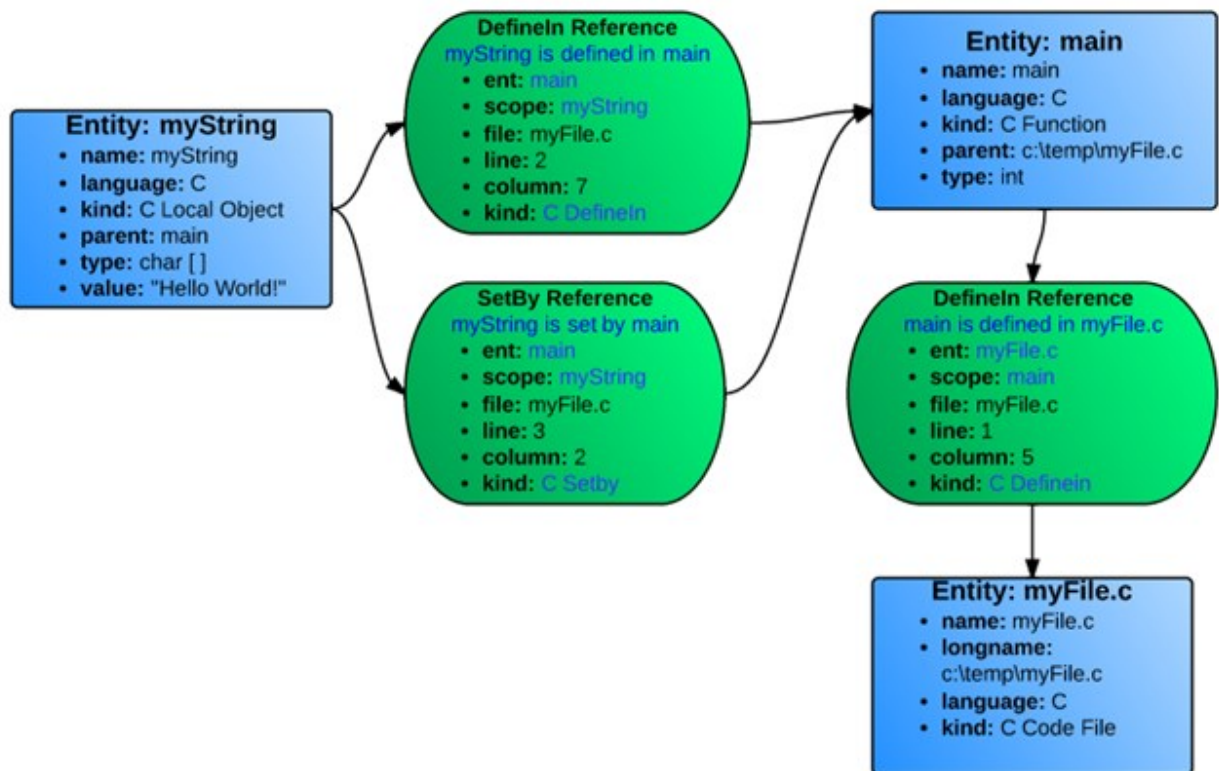
```
void main(){
    char myString[];
    myString = "Hello World!";
}
```

Understand سه موجودیت (آبی) و سه مرجع (سبز) را مشخص می‌کند که در شکل ۵.۱ نشان داده شده است.



شکل ۵.۱ درک ساختار داده برای یک کد C ساده

از آنجایی که همه مراجع روابط بین دو شی هستند، مراجع در واقع در هر دو جهت ذخیره می شوند. از این رو، هر نوع مرجع مخالفی دارد: "Define"، "Set"، "DefineIn"، "Call" و "SetBy"، و "Callby"، و غیره. همان موجودیت های شکل ۵.۱ با مراجع معکوس آنها در شکل ۵.۲ نشان داده شده است.



شکل ۵.۲ مراجع شکل ۱.۵ در جهت معکوس

۵-۱-۱. انواع موجودیت

فهرست جامعی از انواع موجودیت های OpenUnderstand برای زبان برنامه نویسی جاوا را می توان در صفحه Entity kind یافت.

۵-۱-۲. انواع مرجع

فهرست جامعی از انواع مرجع OpenUnderstand برای زبان برنامه نویسی جاوا را می توان در صفحه نوع مرجع یافت.

۵-۲. طرحواره پایگاه داده ERD

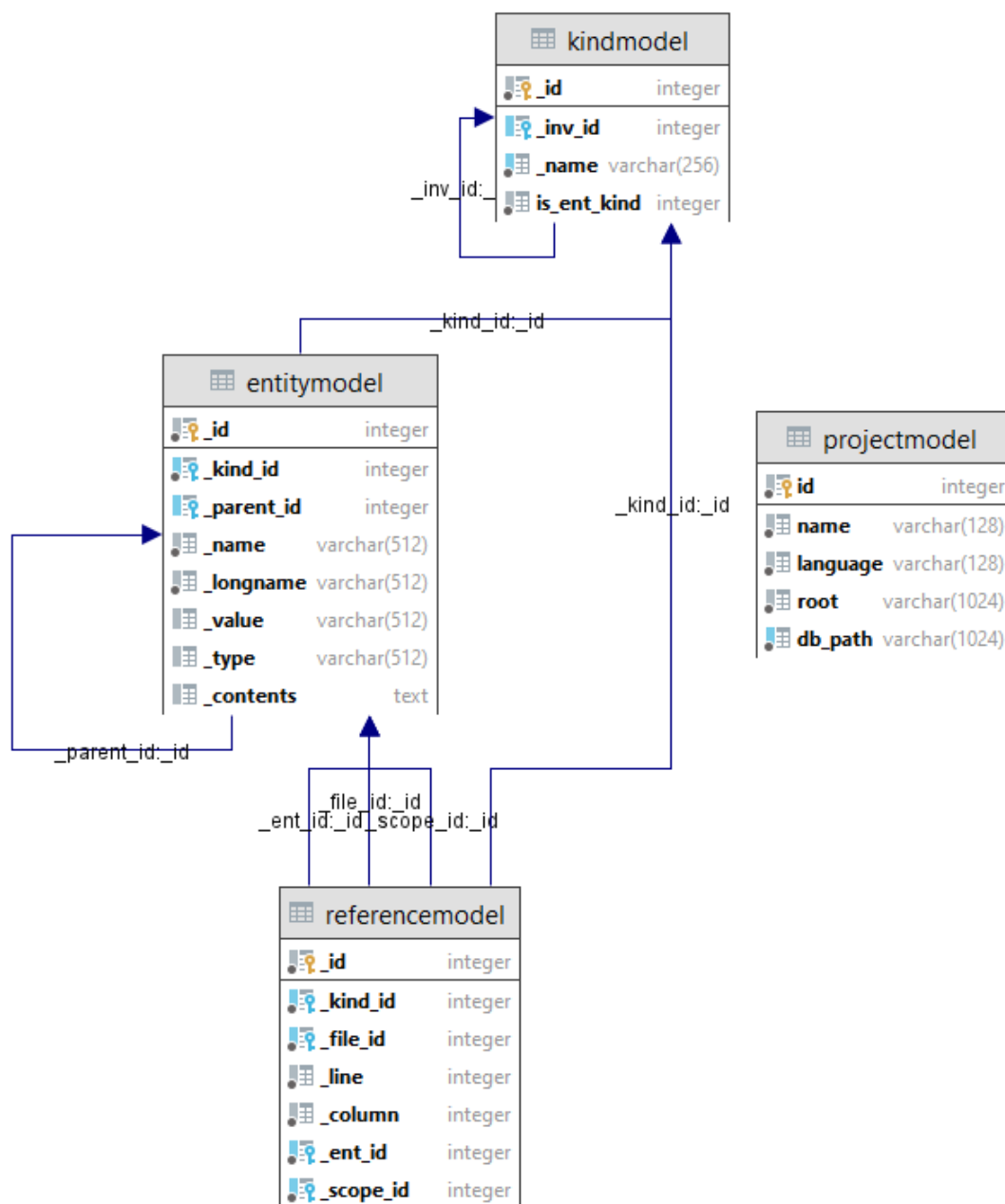
برای بخش پایگاه داده این پروژه از کتابخانه peewee و SQLite ۳ استفاده شده است. خواندن اسناد peewee نیز ضروری است.

نمودار (entity-relationship (ERD برای پایگاه داده طراحی شده برای جدول نماد OpneUnderstand در شکل ۵.۳ نشان داده شده است. چهار جدول مهم در پایگاه داده OpenUnderstand وجود دارد که برای هر پروژه در طول تجزیه و تحلیل استاتیک ایجاد شده است:

Project: برای ذخیره برخی اطلاعات اولیه در مورد پروژه تحت تجزیه و تحلیل مانند نام پروژه، زبان های برنامه نویسی و غیره. این جدول به طور خودکار پر می شود.

Kind: برای ذخیره هر دو نوع موجودیت و مرجع که با is_ent_kind boolean قابل تفکیک هستند. این جدول به طور خودکار پر می شود.

Entity: برای ذخیره موجودیت های جاوا در پروژه. این جدول در طول تجزیه و تحلیل استاتیک برنامه توسط شنوندگان ANTLR پر می شود.



شکل ۵.۳ طرحواره پایگاه داده OpenUnderstand ERD

مرجع: برای ذخیره مراجع جاوا در پروژه. این جدول در طول تجزیه و تحلیل استاتیک برنامه توسط شنوندگان ANTLR پر می شود.

۶. یک مثال کلی که نرم افزار openunderstand چگونه کار می کند

در مثال زیر معیار نرم افزاری تعداد متغیر های integer محاسبه می شود.

```
import sqlite3
import antlr3
from antlr3 import *
from antlr3.tree.Tree import ParseTreeWalker
from antlr_files.JavaLexer import JavaLexer
from antlr_files.JavaParser import JavaParser
from antlr_files.JavaParserListener import JavaParserListener
from peewee import *

db = SQLiteDatabase('symbol_table.db') # Create or connect to
the SQLite database

class SymbolTable(Model):
    name = CharField()
    type = CharField()
    line = IntegerField()

    class Meta:
        database = db

class SymbolTableListener(JavaParserListener):
    def enterVariableDeclaratorId(self, ctx):
        name = ctx.getText()
        symbol_type = ctx.parentCtx.getChild(0).getText()
        line = ctx.start.line

        SymbolTable.create(name=name, type=symbol_type,
line=line) # Store the symbol table entry in the database

def create_symbol_table(input_file):
    lexer = JavaLexer(FileStream(input_file))
    stream = CommonTokenStream(lexer)
    parser = JavaParser(stream)
    tree = parser.compilationUnit()

    listener = SymbolTableListener()
    walker = ParseTreeWalker()
    walker.walk(listener, tree)
```

```
def calculate_java_metric():
    # Perform the necessary calculations using the symbol table
    data stored in the SQLite database
    # Example metric calculation:
    num_variables = SymbolTable.select().where(SymbolTable.type
    == 'int').count()
    print(f"Number of int variables: {num_variables}")

def main():
    db.connect()
    db.create_tables([SymbolTable])

    input_file = 'YourJavaFile.java' # Specify the path to your
    Java file

    create_symbol_table(input_file)
    calculate_java_metric()

    db.close()

if __name__ == '__main__':
    main()
```

۷. نحوه راه اندازی نرم افزار openunderstand

- ابتدا دستور زیر را در ترمینال سیستم خود وارد نمایید:

git clone [git@github.com:m-zakeri/OpenUnderstand.git](https://github.com:m-zakeri/OpenUnderstand.git)

- سپس دستور زیر را وارد نمایید تا تمامی برنج های برنامه fetch شوند.

git fetch

- پس از آن از دستور زیر استفاده کنید تا به برنج dev تغییر وضعیت دهید.

git checkout origin dev

- سپس یک virtualenv ساخته و آن را activate کنید.
 - پس از آن با دستور `pip install -r requirements.txt` کتابخانه های مورد نیاز را نصب کنید.
- حال نرم افزار آماده اجرا می باشد.

برای ساختن جدول نمادها در قسمت root نرم افزار یک فایل به نام `test_openunderstand.py`

می سازیم و کد زیر را در آن قرار می دهیم.

```
import sys
from os import getcwd
from os.path import join
sys.path.append(join(getcwd(), "openunderstand"))
sys.path.append(join(getcwd(), "openunderstand", "oudb"))
sys.path.append(join(getcwd(), "openunderstand", "utils"))
from openunderstand.ouderstand.openunderstand import *

start_parsing(
    repo_address=join(getcwd(), "benchmark", "JSON"),
    db_address=getcwd(),
    db_name="mydb.ldb",
    engine_core="Python3",
    log_address=join(getcwd(), "app.log")
)
```

پس از اجرا در قسمت root برنامه یک فایل به نام app.log ایجاد می شود که دارای محتوای زیر است.

```
2023-11-18 01:01:21,282 - INFO - file parse success
2023-11-18 01:01:21,283 - INFO - file parse success
2023-11-18 01:01:21,283 - INFO - file parse success
2023-11-18 01:01:21,283 - INFO - file parse success
2023-11-18 01:01:21,614 - INFO - The function 'parser' with file address 'JSONString.java' took 1.30 seconds to exe
2023-11-18 01:01:21,283 - INFO - file parse success
2023-11-18 01:01:21,614 - INFO - The function 'parser' with file address 'JSONString.java' took 1.30 seconds to exe
2023-11-18 01:01:21,347 - INFO - file parse success
2023-11-18 01:01:21,614 - INFO - The function 'parser' with file address 'HTTPTokener.java' took 1.30 seconds to ex
2023-11-18 01:01:21,614 - INFO - The function 'parser' with file address 'HTTPTokener.java' took 1.30 seconds to ex
2023-11-18 01:01:21,347 - INFO - file parse success
2023-11-18 01:01:21,615 - INFO - The function 'parser' with file address 'HTTP.java' took 1.30 seconds to execute.
2023-11-18 01:01:21,615 - INFO - The function 'parser' with file address 'HTTP.java' took 1.30 seconds to execute.
2023-11-18 01:01:21,282 - INFO - file parse success
2023-11-18 01:01:21,617 - INFO - The function 'parser' with file address 'XMLXsiTypeConverter.java' took 1.30 secon
2023-11-18 01:01:21,617 - INFO - The function 'parser' with file address 'XMLXsiTypeConverter.java' took 1.30 secon
2023-11-18 01:01:21,621 - INFO - The function 'entity_gen' with file address 'JSONString.java' took 0.01 seconds to
2023-11-18 01:01:21,621 - INFO - The function 'entity_gen' with file address 'JSONString.java' took 0.01 seconds to
```

شکل ۷.۱ نمایی از خروجی فایل لاگ نرم افزار openunderstand

همچنین یک پایگاه داده به نام mydb.udb ساخته می شود که دارای چهار جدول است که جدول entitymodel آن دارای مقادیر زیر است.


Database Structure

Browse Data

Edit Pragma

Execute SQL

Table: entitymodel



Filter in any column

	_id	_kind_id	_parent_id	_name	_longname	_value	_typ
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	NULL	JSONString.java	/home/y/Desktop/iust/OpenUnderstand/...	NULL	NULL
2	2	72	NULL	org	org	NULL	NULL
3	3	72	2	json	org.json	NULL	NULL
4	4	72	1	json	org.json	NULL	NULL
5	5	119	1	JSONString	org.json.JSONString		
6	6	39	5	toJsonString	org.json.JSONString.toJsonString		String
7	7	72	1	org	org	NULL	NULL
8	8	1	NULL	XMLXsiTypeConverter.java	/home/y/Desktop/iust/OpenUnderstand/...	NULL	NULL
9	9	224	NULL	String	org.json.String	NULL	NULL
10	10	225	NULL	value	org.json.value	NULL	NULL
11	11	72	8	json	org.json	NULL	NULL
12	12	123	8	XMLXsiTypeConverter	org.json.XMLXsiTypeConverter		
13	13	33	12	convert	org.json.XMLXsiTypeConverter.convert		T
14	14	75	13	value	org.json.XMLXsiTypeConverter.convert.value		String
15	15	72	8	org	org	NULL	NULL
16	16	1	NULL	HTTPTokener.java	/home/y/Desktop/iust/OpenUnderstand/...	NULL	NULL
17	17	39	16	nextToken	org.json.HTTPTokener.nextToken	NULL	String
18	18	84	NULL	StringBuilder	StringBuilder	NULL	NULL
19	19	225	NULL	string	org.json.string	NULL	NULL
20	20	225	NULL	nextToken	org.json.nextToken	NULL	NULL
21	21	224	NULL	char	org.json.char	NULL	NULL
22	22	225	NULL	c	org.json.c	NULL	NULL
23	23	225	NULL	q	org.json.q	NULL	NULL
24	24	1	NULL	JSONStringer.java	/home/y/Desktop/iust/OpenUnderstand/...	NULL	NULL
25	25	224	NULL	StringBuilder	org.json.StringBuilder	NULL	NULL
26	26	225	NULL	sb	org.json.sb	NULL	NULL
27	27	39	24	JSONStringer	org.json.JSONStringer.JSONStringer	NULL	
28	28	72	16	json	org.json	NULL	NULL

شکل ۷.۲ نمایشی از داده های جدول موجودیت در پایگاه داده ساخته شده بوسیله نرم افزار openunderstand

جدول kind model آن دارای مقادیر زیر است

Table: kindmodel Filter in any column

	id	inv_id	_name	is_ent_kind
	Fi...	Filter	Filter	Filter
1	1	NULL	Java File	1
2	2	NULL	Java File Jar	1
3	3	NULL	Java Method Constructor Member Default	1
4	4	NULL	Java Method Constructor Member Protected	1
5	5	NULL	Java Method Constructor Member Private	1
6	6	NULL	Java Method Constructor Member Public	1
7	7	NULL	Java Static Final Method Default Member	1
8	8	NULL	Java Static Final Method Private Member	1
9	9	NULL	Java Static Final Method Protected Member	1
10	10	NULL	Java Static Final Method Public Member	1
11	11	NULL	Java Static Final Generic Method Default ...	1
12	12	NULL	Java Static Final Generic Method Private ...	1
13	13	NULL	Java Static Final Generic Method Protected...	1
14	14	NULL	Java Final Method Default Member	1
15	15	NULL	Java Static Final Generic Method Public ...	1
16	16	NULL	Java Final Method Private Member	1
17	17	NULL	Java Final Method Protected Member	1
18	18	NULL	Java Final Method Public Member	1
19	19	NULL	Java Generic Final Method Default Member	1
20	20	NULL	Java Generic Final Method Private Member	1
21	21	NULL	Java Static Method Default Member	1
22	22	NULL	Java Final Generic Method Protected ...	1
23	23	NULL	Java Static Method Private Member	1
24	24	NULL	Java Final Generic Method Public Member	1
25	25	NULL	Java Static Method Protected Member	1
26	26	NULL	Java Static Method Public Member	1
27	27	NULL	Java Static Generic Method Default Member	1
28	28	NULL	Java Static Generic Method Private Member	1

شکل ۷.۳ نمایی از داده های جدول انواع در پایگاه داده ساخته شده بوسیله نرم افزار openunderstand

جدول projectmodel آن دارای مقادیر زیر است

Table: projectmodel

id	name	language	root	db_path
1	1	JSON	Java	/home/y/Desktop/iust/OpenUnderstand/...

شکل ۷.۴ نمایشی از داده های جدول پروژه ها در پایگاه داده ساخته شده بوسیله نرم افزار openunderstand

و جدول referencemodel آن دارای مقادیر زیر می باشد.

Table: referencemodel							
	_id	_kind_id	_file_id	_line	_column	_ent_id	_scope_id
	Filter	Filter	Filter	Filt...	Filter	Filter	Filter
1	1	194	1	1	0	1	4
2	2	195	1	1	0	4	1
3	3	194	1	35	7	1	5
4	4	195	1	35	7	5	1
5	5	194	1	42	11	5	6
6	6	195	1	42	11	6	5
7	7	192	1	1	8	7	1
8	8	193	1	1	8	1	7
9	9	192	1	1	12	4	7
10	10	193	1	1	12	7	4
11	11	224	10	65	14	9	10
12	12	225	9	65	21	10	9
13	13	194	8	1	0	8	11
14	14	195	8	1	0	11	8
15	15	194	8	64	7	8	12
16	16	195	8	64	7	12	8
17	17	194	8	65	4	12	13
18	18	195	8	65	4	13	12
19	19	194	8	65	14	13	14
20	20	195	8	65	14	14	13
21	21	192	8	1	8	15	8
22	22	193	8	1	8	8	15
23	23	192	8	1	12	11	15
24	24	193	8	1	12	15	11
25	25	190	16	52	27	18	17
26	26	191	16	52	27	17	18
27	27	224	19	39	23	9	19
28	28	225	0	30	20	10	0

شکل ۷.۵. نمایشی از داده های جدول مراجع در پایگاه داده ساخته شده بوسیله نرم افزار openunderstand

حال برای فراخوانی api ها یک فایل دیگر به نام test_api.py میسازیم و کد زیر را در آن قرار می دهیم.

```
import sys
from os import getcwd
from os.path import join
sys.path.append(join(getcwd(), "openunderstand"))
sys.path.append(join(getcwd(), "openunderstand", "oudb"))
sys.path.append(join(getcwd(), "openunderstand", "utils"))

import openunderstand.ouderstand as und

_db = und.open("/home/y/Desktop/iust/OpenUnderstand/mydb.udb")

print(
    len(
        _db.ents("class")
    )
)
```

```
)  
)
```

که خروجی آن عدد ۴۳۴ می باشد.

توجه داشته باشید که آرگومان ورودی `und.open()` باید برابر مکان قرار دادن پایگاه داده شما باشد.

۸. اضافه کردن type به جدول نماد ها

در قسمت `OpenUnderstand/openunderstand/ounderstand/listener_and_parser.py`

ابتدا یک listener برای `extend` می نویسیم.

```
from gen.javaLabeled.JavaParserLabeledListener import  
JavaParserLabeledListener  
from gen.javaLabeled.JavaParserLabeled import JavaParserLabeled  
import analysis_passes.class_properties as class_properties  
  
class ExtendCoupleAndExtendCoupleBy(JavaParserLabeledListener):  
    """  
    # TODO: Implementing the ANTLR listener pass for Java Call and  
    Java Callby reference kind  
    """  
    def __init__(self):  
        self.implement = []  
  
    def enterClassDeclaration(self, ctx:  
        JavaParserLabeled.ClassDeclarationContext):  
  
        # if ctx.IMPLEMENTS():  
        scope_parents =  
        class_properties.ClassPropertiesListener.findParents(ctx)  
  
        if len(scope_parents) == 1:  
            scope_longname = scope_parents[0]  
        else:  
            scope_longname = ".".join(scope_parents)  
  
        line = ctx.children[0].symbol.line  
        col = ctx.children[0].symbol.column  
        if ctx.EXTENDS():  
            extendedBy =
```

```
ctx.typeType().classOrInterfaceType().IDENTIFIER(i=۰)
    print("[DEBUG] ExtendCouples: ", scope_parents,
scope_longname, extendedBy)

    self.implement.append(
        {
            "scope_kind": "Class",
            "scope_name": ctx.IDENTIFIER().__str__(),
            "scope_longname": str(scope_longname),
            "scope_parent": scope_parents[-۲]
            if len(scope_parents) > ۲
            else None,
            "scope_contents": ctx.getText(),
            "scope_modifiers":
class_properties.ClassPropertiesListener.findClassOrInterfaceMod
ifiers(
                ctx
            ),
            "line": line,
            "col": col,
            "type_ent_longname": str(extendedBy),
        }
    )
```

۸-۱. ذخیره کردن مدل ها

برای ذخیره کردن entity model ها به همراه reference model ها تابع زیر را در کلاس project در آدرس `OpenUnderstand/openunderstand/ounderstand/project.py` قرار می دهیم.

```
def addExtendCoupleOrExtendCoupleByRefs(self, ref_dicts,
file_ent, file_address):
    for ref_dict in ref_dicts:
        scope = EntityModel.get_or_create(
            _kind=self.findKindWithKeywords(
                ref_dict["scope_kind"],
ref_dict["scope_modifiers"]
            ),
            _name=ref_dict["scope_name"],
            _parent=ref_dict["scope_parent"]
            if ref_dict["scope_parent"] is not None
            else file_ent,
            _longname=ref_dict["scope_longname"],
```

```

        _contents=ref_dict["scope_contents"],
    )[0]
    ent = self.getImplementEntity(
        ref_dict["type_ent_longname"], file_address, file_ent
    )
    extend_ref = ReferenceModel.get_or_create(
        _kind=178,
        _file=file_ent,
        _line=ref_dict["line"],
        _column=ref_dict["col"],
        _ent=ent,
        _scope=scope,
    )
    extendBy_ref = ReferenceModel.get_or_create(
        _kind=179,
        _file=file_ent,
        _line=ref_dict["line"],
        _column=ref_dict["col"],
        _ent=scope,
        _scope=ent,
    )

```

در کلاس ListenersAndParsers یک تابع به نام extend_coupled_listener اضافه می کنیم.

```

@timer_decorator()
def extend_coupled_listener(self, tree, file_ent, file_address,
p):
    try:
        listener = ExtendCoupleAndExtendCoupleBy()
        p.Walk(listener, tree)
        p.addExtendCoupleOrExtendCoupleByRefs(listener.implement,
file_ent, file_address)
        self.logger.info("extends coupled refs success ")
    except Exception as e:
        self.logger.error(
            "An Error occurred in file extends coupled refs : " +
file_address + "\n" + str(
                e)
        )

```

سپس	این	تابع	listener	را	در	قسمت
						OpenUnderstand/openunderstand/ounderstand/parsing_proccess.py در متد زیر به لیست
						listeners اضافه می کنیم.

```
def process_file(file_address):
    p = Project()
    lap = ListenersAndParsers()
    tree, parse_tree, file_ent =
lap.parser(file_address=file_address, p=p)
    if tree is None and parse_tree is None and file_ent is None:
        return
    entity_generator = lap.entity_gen(file_address=file_address,
parse_tree=parse_tree)
    listeners = [
        lap.create_listener,
        lap.type_listener,
        lap.define_listener,
        lap.declare_listener,
        lap.override_listener,
        lap.callby_listener,
        lap.couple_listener,
        lap.useby_listener,
        lap.setby_listener,
        lap.dotref_listener,
        lap.throw_listener,
        lap.extend_coupled_listener,
    ]
    lap.modify_listener(
        entity_generator=entity_generator,
        parse_tree=parse_tree,
        file_address=file_address,
        p=p,
    )
    for listener in listeners:
        listener(file_address=file_address, p=p,
file_ent=file_ent, tree=tree)
```

۹. اضافه کردن api معیار^۳

در قسمت `OpenUnderstand/openunderstand/oudb/api.py` دو قسمت برای اضافه کردن api معیار

تعبیه شده که در قسمت زیر آنها را مشاهده می فرمایید.

```
def metric(self, metriclist): # real signature unknown;
    restored from __doc__
    """
    ent.metric(metriclist) -> dict key=string value=metricvalue

    Return the metric value for each item in metriclist

    Metric list must be a tuple or list containing the names of
    metrics
    as strings. If the metric is not available, it's value will
    be None.
    """
    return {}

def metrics(self): # real signature unknown; restored from
    __doc__
    """
    ent.metrics() -> list of strings

    Return a list of metric names defined for the entity.
    """
    return []
```

حال می خواهیم برای نمونه یک api معیار نرم افزاری به آن اضافه کرده و از آن استفاده کنیم.

ابتدا نمونه کد زیر را برای خود understand اجرا می کنیم.

```
def check_metrics(self, my_path:str=""):
    _db = understand.open(my_path)
    und_all_results = {}
    for ent in _db.ents("Class"):
        ent_name = ent.name()
        print(ent.metric(["CountDeclMethodAll"]))
```

^۳ Metric

```
all_methods = ent.metric(["CountDeclMethodAll"]).get(
    "CountDeclMethodAll", ۰)
und_all_results[ent_name] = all_methods

print(und_all_results)
```

که به خروجی زیر می‌رسیم:

```
{'CountDeclMethodAll': ۰}
{'CountDeclMethodAll': ۰}
{'CountDeclMethodAll': ۰}
{'CountDeclMethodAll': ۰}
{'CountDeclMethodAll': ۰}
{'CountDeclMethodAll': ۱۰۶}
{'CountDeclMethodAll': ۵۱}
{'CountDeclMethodAll': ۱۳۰}
{'CountDeclMethodAll': ۳۶}
{'CountDeclMethodAll': ۳۶}
...
```

سپس شروع به پیاده سازی کد در قسمت metrics می‌کنیم ، ابتدا یک فایل به نام count_decl_method_all.py می‌سازیم.

در این قسمت مقادیر CountDeclMethodAll را محاسبه می‌نماییم. که همان مقدار تابع های یک کلاس به شامل کلاس های ارث بری شده از آن می‌باشد.

کد زیر شامل پیاده سازی این معیار می‌باشد.

```
from oudb.models import EntityModel, KindModel, ReferenceModel
from utils.utilities import setup_logger
```

```

logger = setup_logger()

def count_decl_method_all(ent_model = None) -> int:
    number_of_methods = 0
    class_methods = {}
    files = []
    extends_class_names = {}
    kinds = KindModel.select().where(
        KindModel._name.contains("Extend")
    )
    refs =
ReferenceModel.select().where(ReferenceModel._kind_id.in_(kinds)
)
    for e in
EntityModel.select().where(EntityModel._id.in_(refs)):
        extends_class_names.update({e._longname: e._name})
        if ent_model.kind() == 1:
            files.append(ent_model._longname)
        if "Class" in ent_model.kind().name():
            class_methods[ent_model._name] = 0
            # get class methods number
        for ent_model in EntityModel.select():
            try:
                if "Method" in ent_model._kind._name:
                    exists =
class_methods.get(ent_model._parent._name, -1)
                    if exists == -1:
                        class_methods[ent_model._parent._name] = 0
                    else:
                        class_methods[ent_model._parent._name] += 1
            except Exception as e:
                logger.error(f"error to calculate
count_decl_method_all metric in {ent_model._kind._name} kind")

        for cm in class_methods:
            visited = []
            temp = cm
            while extends_class_names.__contains__(temp):
                t = extends_class_names[temp]
                if not visited.__contains__(t):
                    visited.append(t)
                    temp = "-۹۹۹۹"

            for v in visited:

```



```

        number_of_methods += class_methods[v]
    return number_of_methods

```

پس از پیاده سازی کد را به صورت زیر به api متریک ها اضافه می کنیم.

```

def metric(self, metric_list: list = None) -> dict: # real
signature unknown; restored from __doc__
    """
    ent.metric(metriclist) -> dict key=string value=metricvalue

    Return the metric value for each item in metriclist

    Metric list must be a tuple or list containing the names of
metrics
as strings. If the metric is not available, it's value will
be None.
    """
    metrics = {}
    for item in metric_list:
        if item not in self.metrics():
            raise ValueError(f"metric {item} is not in metric
list")
    for item in metric_list:
        if item == "CountDeclMethodAll":
            metrics.update({"CountDeclMethodAll":
count_decl_method_all(self)})
    return metrics

def metrics(self): # real signature unknown; restored from
__doc__
    """
    ent.metrics() -> list of strings

    Return a list of metric names defined for the entity.
    """
    return ["CountDeclMethodAll"]

```

توجه کد بالا بدلیل پیاده سازی نکردن listener های زیر

.	Java Extend Couple External		۱۸۲
---	-----------------------------	--	-----

•	Java Extend Coupleby External	۱۸۲	۱۸۳
•	Java Extend Couple Implicit External		۱۸۴
•	Java Extend Coupleby Implicit External	۱۸۴	۱۸۵
•	Java Extend Couple Implicit		۱۸۶
•	Java Extend Coupleby Implicit	۱۸۶	۱۸۷

بدرستی کار نمی کند برای تمرین اول لیسنر های زیر را به کد اضافه کرده و سپس دوباره کد را اجرا کنید و تفاوت خروجی را گزارش دهید.

از قطعه کد زیر که در قسمت root برنامه استفاده شده برای اجرای معیار نوشته شده استفاده نمایید.

```
import sys
from os import getcwd
from os.path import join
sys.path.append(join(getcwd(), "openunderstand"))
sys.path.append(join(getcwd(), "openunderstand", "oudb"))
sys.path.append(join(getcwd(), "openunderstand", "utils"))
sys.path.append(join(getcwd(), "openunderstand", "metrics"))
import openunderstand.ouderstand as und

_db = und.open("/home/y/Desktop/iust/OpenUnderstand/mydb.ldb")

und_all_results = {}
for ent in _db.ents("Class"):
    ent_name = ent.name()
    print(ent.metric(["CountDeclMethodAll"]))
    all_methods =
ent.metric(["CountDeclMethodAll"]).get("CountDeclMethodAll", .)
```

```
und_all_results[ent_name] = all_methods
print(und_all_results)
```

توجه: کد ممکن است در برای اجرا در ویندوز دچار مشکل شود برای این مشکل توجه داشته باشید که مقادیر / را به \

تغییر دهید و همچنین در فایل /OpenUnderstand/openunderstand/ounderstand/runner.py/ را از شکل

زیر

```
from multiprocessing import cpu_count, Pool
from ounderstand.parsing_process import process_file, get_files

def runner(path_project: str = ""):
    files = get_files(path_project)
    with Pool(cpu_count()) as pool:
        pool.map_async(process_file, files)
        pool.close()
        pool.join()
```

به شکل زیر تغییر دهید:

```
# from multiprocessing import cpu_count, Pool
from ounderstand.parsing_process import process_file, get_files

def runner(path_project: str = ""):
    files = get_files(path_project)
    for item in files:
        process_file(item)
    # with Pool(cpu_count()) as pool:
    #     pool.map_async(process_file, files)
    #     pool.close()
    #     pool.join()
```

۱۰. لینک های مفید

<https://documentation.scitools.com/html/python/index.html>

<https://github.com/m-zakeri/OpenUnderstand/tree/dev/openunderstand>