



Compiler project document

Group 10

Group members:

Ali Sedaghi

Mohammad Yarmoghadam

Amir Masoud Golestane

Hasti Karamdel

Introduction	3
What is Understand?	3
This project.....	3
Our sections	4
Project division	5
What is expected?	5
Implementation	7
Initializing	7
Import/Importby	12
Modify (Deref) Partial and Modifyby (Deref) Partial.....	18
Tests and results.....	24
Procedure and Challenges.....	34
Conclusions and Recommendations	36
Additional Sources	37

Introduction

What is Understand?

Understand is a static analysis tool focused on source code comprehension, metrics, and standards testing. It is designed to help maintain and understand large amounts of legacy or newly created source code. It provides a cross-platform, multi-language, maintenance-oriented IDE (interactive development environment).

Understand uses more than 50 different graphs to help you visualize exactly what your code is doing and how it is built. Browse call trees, explore dependencies, verify UML structures or design your own graphs with the API.

Understand has architecture features that help you create hierarchical aggregations of source code units. You can name these units and manipulate them in various ways to create interesting hierarchies for analysis.

This project

Unfortunately, the Understand API source code is not publicly available, making it difficult to change, customize, and reuse in new activities and environments which appears in academic researches.

This project aims to provide an open-source implementation of the Understand Python API to analyze the source codes. We primarily focus on implementing the API for Java programs using Python programming languages and compiler tools such as ANTLR. To develop an open-source implementation of Understand Python API, we look at the structures used by Understand for analyzing source codes.

Our sections

Most of the data captured by Understand involves Entities and References.

Entity: An Entity is anything in the code that Understand captures information on: i.e., A file, a class, a variable, a function, etc. In the Perl API, Entities are represented with the `Understand::Ent` class. In Python, it is the `Understand.Ent` class.

Reference: A specific place where an entity appears in the code. A reference is always defined as a relationship between two entities. e.g., function Bar is called on line 14 of function Foo. In the Perl API, References are represented with the `Understand::Ref` class. In Python, it is the `Understand.Ref` class.

Every entity and reference have a unique set of attributes that can be queried by the API. A few of the attributes you can view for an entity would be its name, its type, any associated comments, what kind of entity it is, and if it has them: its parent entity and its parameters. On the other hand, a reference would have both of the entities associated with it as well as the file, line, and column where the reference occurs and what kind of reference it is.

This project should support different reference kinds that we can have in our java code, each of these references can also use different entities. This is the table of the reference kinds which are each implemented by a different group.

You can see the reference kinds in this table:

https://m-zakeri.github.io/OpenUnderstand/reference_kinds/

You can see the entity kinds in this table:

https://m-zakeri.github.io/OpenUnderstand/entity_kinds/

Our group has tried to implement a code for the entities of Import/Import by & Modify (Deref) Partial and Modifyby (Deref) Partial. To recognize these entities, we have to use different entity kinds to Analyze our java code and find the references of each time they are used.

Project division

We decided it is best to divide the project into different parts we could each do separately and also a part we could all participate together. Therefore two of us worked on import/import by and the two others worked on modify/modify by. In the end we all gathered to study and test our codes and write this documentation.

What is expected?

We need to implement a standalone python code which scans through a java project. Then it must be able to detect the reference kind we want (import or modify) and fill our database with the attributes we need for each kind.

Import/ImportBy:

Java Import Demand indicates a file has an on-demand import statement for a package or class. For example, if a file named file.java has imported a class, we would store the following data in our table:

- File id
- Imported class
- Entity kind id
- Line and Col in which import is used
- And also the responding entity which is imported by the importing entity

Reference:

https://m-zakeri.github.io/OpenUnderstand/reference_kinds/#java-import-and-importby

Modify (Deref) Partial and Modifyby (Deref) Partial:

Java Modify and Modifyby indicates that a variable's value is modified or both read and set, as with the increment (++), decrement (--), and assignment/operator combinations (*=, /=, ...). This reference is like Modify and Modifyby. But, it is used when an entity modifies some elements of a variable of collection type. Same as import we also need to detect in which scope a variable is being modified, for example if it has been modified in a function, we would add the following data to our table:

- File id
- Scope id
- Entity kind id
- Line and Col in which import is used
- And also the responding entity which is modified by the modifying entity

Reference:

https://m-zakeri.github.io/OpenUnderstand/reference_kinds/#java-modify-and-modifyby

All of the implementation codes can be found here:

<https://github.com/mohammadym/OpenUnderstand>

Implementation

Initializing

This section of the code is repeated in both of our files, because we implemented a standalone python code for each of the commands.

Imports:

First, we need to import antlr, Java Lexer, Parser and Listener. Then to connect and use the database we need to import the API and functions needed to fill the tables, the models for entity kinds and etc.

```
import os
from antlr4 import *
from pathlib import Path
from gen.javaLabeled.JavaLexer import JavaLexer
from gen.javaLabeled.JavaParserLabeled import JavaParserLabeled
from gen.javaLabeled.JavaParserLabeledListener import JavaParserLabeledListener
from oudb.fill import main as db_fill
from oudb.api import create_db, open as db_open
from oudb.models import KindModel, EntityModel, ReferenceModel
```

Benchmark settings:

To run and test each of the benchmark projects we need to set a project index and add the names of the projects that we need to be tested. We also need to add a path so that the project files can be read and tested from there, to handle this matter we have made a similar path for each of the projects and the only thing that differs each path is the project name itself; resulting an array for the paths of each project. We also need to set the path of our database file which includes the final results and tables.

```
PRJ_INDEX = 3
REF_NAME = "import"
```

```

def get_project_info(index, ref_name):
    project_names = [
        'calculator_app',
        'JSON',
        'testing_legacy_code',
        'jhotdraw-develop',
        'xerces2j',
        'jvlt-1.3.2',
        'jfreechart',
        'ganttproject',
        '105_freemind',
    ]
    project_name = project_names[index]
    db_path = f"../databases/{ref_name}/{project_name}"
    if ref_name == "origin":
        db_path = db_path + ".udb"
    else:
        db_path = db_path + ".oudb"
    project_path = f"../benchmarks/{project_name}"

    db_path = os.path.abspath(db_path)
    project_path = os.path.abspath(project_path)

    return {
        'PROJECT_NAME': project_name,
        'DB_PATH': db_path,
        'PROJECT_PATH': project_path,
    }

```

Init:

In the beginning of the class, we set the database name, project name and directory and also the two arrays with the names and paths of the files.

```

def __init__(self, db_name, project_dir, project_name=None):
    self.db_name = db_name
    self.project_dir = project_dir

```



```
self.project_name = project_name
self.files = []
```

Initdb:

We used createdb to create our database then used the fill function to add the models, the database is added to the path we gave it in the beginning and is ready to use.

```
def init_db(self):
    create_db(self.db_name, self.project_dir, self.project_name)
    db_fill()
    db_open(self.db_name)
```

Get Java Files:

The project directories that we have include many files, that is why we have to find the .java files in order to continue. For each java file we add the name and path to the arrays we have.

```
def get_java_files(self):
    for dir_path, _, file_names in os.walk(self.project_dir):
        for file in file_names:
            if '.java' in str(file):
                path = os.path.join(dir_path, file)
                path = path.replace("/", "\\")
                path = os.path.abspath(path)
                self.files.append((file, path))
                add_java_file_entity(path, file)
```

Get Parent:

This function checks our database for the parent entity if there is a parent entity with the id of the “java file” entity kind and also the name and the path given, it will return its object.

```
def get_parent(parent_file_name, files):
    file_names, file_paths = zip(*files)
    parent_file_index = file_names.index(parent_file_name)
    parent_file_path = file_paths[parent_file_index]
    parent_entity = EntityModel.get_or_none(
        _kind=KindModel.get_or_none(_name="Java File").get_id(),
        _name=parent_file_name,
        _longname=parent_file_path,
    )
    return parent_entity, parent_file_path
```

Add Java file entity:

This function gets each java file name and path from the arrays and sets its entity in the entity kinds table, or if there already is an id for that entity, it will return the object.

```
def add_java_file_entity(file_path, file_name):
    kind_id = KindModel.get_or_none(_name="Java File").get_id()
    obj, _ = EntityModel.get_or_create(
        _kind=kind_id,
        _name=file_name,
        _longname=file_path,
        _contents=FileStream(file_path, encoding="utf-8"),
    )
    return obj
```

Get parse tree:

This function gets the file from the file path given(using fileStream) then creates a java lexer for that file. From that lexer we can tokenize it and make our parser.

```
def get_parse_tree(file_path):  
    file = FileStream(file_path, encoding="utf-8")  
    lexer = JavaLexer(file)  
    tokens = CommonTokenStream(lexer)  
    parser = JavaParserLabeled(tokens)  
    return parser.compilationUnit()
```

Import/Importby

After getting all of the java files and adding the entity kinds to our database table, we need to focus on finding the import references in our files.

Import Listener:

The goal of this class is to create a list of objects for each “import” in our file. The object should have the attributes needed to reference an import. At first we get the Imported classes name and check if it already is in the array of names we created before, because if it isn’t that means our imported class is a java built-in class type. Then with the help of our listener we get the line and column where the import is used in our file. After setting all the attributes we shape our object for that import and add it to the list.

```
class ImportListener(JavaParserLabeledListener):
    def __init__(self, files):
        self.repository = []
        self.files = files

    def enterImportDeclaration(self, ctx: JavaParserLabeled.ImportDeclaration):
        imported_class_longname = ctx.qualifiedName().getText()
        imported_class_name = imported_class_longname.split('.')[-1]

        is_built_in = False
        imported_class_file_name = imported_class_name + ".java"
        if imported_class_file_name not in [file[0] for file in self.files]:
            is_built_in = True
            imported_class_file_name = None

        line = ctx.children[0].symbol.line
        col = ctx.children[0].symbol.column

        self.repository.append({
            'imported_class_name': imported_class_name,
            'imported_class_longname': imported_class_longname,
            'is_built_in': is_built_in,
            'imported_class_file_name': imported_class_file_name,
```

```

        'line': line,
        'column': col,
    })

```

Imported Entity Listener:

Same as the previous class this listener indicates the kind of our imported entity

```

class ImportedEntityListener(JavaParserLabeledListener):
    def __init__(self, name):
        self.body = None
        self.branches = None
        self.type = None
        self.name = name

    def enterClassDeclaration(self, ctx: JavaParserLabeled.ClassDeclarationContext):
        if self.name == ctx.IDENTIFIER().getText():
            self.body = ctx.getText()
            self.branches = ctx.parentCtx.children

    def enterInterfaceDeclaration(self, ctx:
JavaParserLabeled.InterfaceDeclarationContext):
        if self.name == ctx.IDENTIFIER().getText():
            self.body = ctx.getText()
            self.branches = ctx.parentCtx.children

    def enterEnumDeclaration(self, ctx: JavaParserLabeled.EnumDeclarationContext):
        if self.name == ctx.IDENTIFIER().getText():
            self.body = ctx.getText()
            self.branches = ctx.parentCtx.children

```

Add Imported Entity:

At first we will check if the entity is a built in java class the kindid of it should be set to the id of "Java Unknown Class Type Member", also this kind of class doesn't have a parent. If the entity isn't built in we will set its kind using get kind method. We will also get the parent of this entity from the

functions explained. At last we will get the class body to add the contents of that class to our table.

```
def add_imported_entity(i, files):
    if i['is_built_in']:
        imported_entity, _ = EntityModel.get_or_create(
            _kind=KindModel.get_or_none(_name="Java Unknown Class Type
Member").get_id(),
            _parent=None,
            _name=i['imported_class_name'],
            _longname=i['imported_class_longname'],
        )
    else:
        parent_entity, parent_file_path = get_parent(i['imported_class_file_name'], files)
        prefixes, class_body, kind = get_imported_entity(parent_file_path)
        entity_kind = get_kind_name(prefixes, kind)
        imported_entity, _ = EntityModel.get_or_create(
            _kind=KindModel.get_or_none(_name=entity_kind).get_id(),
            _parent=parent_entity.get_id(),
            _name=i['imported_class_name'],
            _longname=i['imported_class_longname'],
            _contents=class_body,
        )
    return imported_entity
```

Get Imported Entity:

This method returns the imported entity attributes. At first we get the parse tree of the path we have, and we also pass the file path to the imported entity listener explained before. After getting our listener, we can start walking in the tree. As we walk in the tree we check if the kind of our branch is class, interface or enum. Therefore set the corresponding attributes and return it.

```
def get_imported_entity(file_path):
    tree = get_parse_tree(file_path)
    listener = ImportedEntityListener(Path(file_path).stem)
    walker = ParseTreeWalker()
```

```

walker.walk(listener=listener, t=tree)

prefixes = ""
kind = ""
for branch in listener.branches:
    if type(branch) == JavaParserLabeled.ClassDeclarationContext:
        kind = "Class"
        break
    elif type(branch) == JavaParserLabeled.InterfaceDeclarationContext:
        kind = "Interface"
        break
    elif type(branch) == JavaParserLabeled.EnumDeclarationContext:
        kind = "Enum Class"
        break
    prefixes += branch.getText() + " "
return prefixes, listener.body, kind

```

Kind Type:

This method helps us figure out the exact kind name for each entity. After checking the prefixes we can assemble them in one string(in the right order) as the final name.

```

def get_kind_name(prefixes, kind):
    p_static = ""
    p_abstract = ""
    p_generic = ""
    p_type = "Type"
    p_visibility = "Default"
    p_member = "Member"

    if "static" in prefixes:
        p_static = "Static"

    if "generic" in prefixes:
        p_generic = "Generic"

    if "abstract" in prefixes:
        p_abstract = "Abstract"

```

```

elif "final" in prefixes:
    p_abstract = "Final"

if "private" in prefixes:
    p_visibility = "Private"
elif "public" in prefixes:
    p_visibility = "Public"
elif "protected" in prefixes:
    p_visibility = "Protected"

if kind == "Interface":
    p_member = ""

if kind == "Method":
    p_type = ""

s = f"Java {p_static} {p_abstract} {p_generic} {kind} {p_type} {p_visibility}
{p_member}"
s = " ".join(s.split())
return s

```

Add references:

This function gets the imported and importing entity and adds them in the final table of our database.

```

def add_references(importing_ent, imported_ent, ref_dict):
    ref, _ = ReferenceModel.get_or_create(
        _kind=KindModel.get_or_none(_name="Java Import").get_id(),
        _file=importing_ent.get_id(),
        _line=ref_dict['line'],
        _column=ref_dict['column'],
        _ent=imported_ent.get_id(),
        _scope=importing_ent.get_id(),
    )
    inverse_ref, _ = ReferenceModel.get_or_create(
        _kind=KindModel.get_or_none(_name="Java Importby").get_id(),
        _file=importing_ent.get_id(),
        _line=ref_dict['line'],

```



```

        _column=ref_dict['column'],
        _ent=importing_ent.get_id(),
        _scope=imported_ent.get_id(),
    )

```

Main:

First we make an instance of the project and call `init_db`. After setting our arrays with the java files found, we need to match the corresponding name and path of each file, then for each file we find the importing entity as explained before. We need to use `get_parse_tree` to get our tree, then use the listener we made for import before so we can walk in our tree. Now we have our repository which is the list with all of our import objects. For each item in that list we need to find the imported entity using the `add_imported_entity` method explained before. At last we have our importing and imported entity and also the matching repository object in the listener so we pass them to `add_references` which adds them to the final references table in our database.

`def main():`

```

    info = get_project_info(PRJ_INDEX, REF_NAME)
    p = Project(info['DB_PATH'], info['PROJECT_PATH'], info['PROJECT_NAME'])
    p.init_db()
    p.get_java_files()

```

`for file_name, file_path in p.files:`

```

    importing_entity = add_java_file_entity(file_path, file_name)

```

```

    tree = get_parse_tree(file_path)
    listener = ImportListener(p.files)
    walker = ParseTreeWalker()
    walker.walk(listener, tree)

```

`for i in listener.repository:`

```

    imported_entity = add_imported_entity(i, p.files)
    add_references(importing_entity, imported_entity, i)

```

Modify (Deref) Partial and Modifyby (Deref) Partial

We have defined most of the initial functions and set ups in the previous sections, in this section we will describe the functions that are specifically designed to find the Modifyderef Partial command and shape our reference table in the database.

Get prefixes:

In order to find the name of our entity kind with get kind name method(explained in the previous parts) we need to have these prefixes.

```
def get_prefixes(ctx, ctx_type):
    branches = ctx.parentCtx.children
    prefixes = ""
    for branch in branches:
        if type(branch).__name__ == ctx_type:
            break
    prefixes += branch.getText() + " "
    return prefixes
```

Class Listener:

This class, is our listener which checks through our file and adds the classes to the database table of entities.

```
class ClassListener(JavaParserLabeledListener):
    def __init__(self, files, file_name):
        self.files = files
        self.file_name = file_name

    def enterClassDeclaration(self, ctx: JavaParserLabeled.ClassDeclarationContext):
        parent_entity, parent_file_path = get_parent(self.file_name, self.files)
        prefixes = get_prefixes(ctx, "ClassDeclarationContext")
        kind_name = get_kind_name(prefixes, kind="Class")
        obj, _ = EntityModel.get_or_create(
            _kind=KindModel.get_or_none(_name=kind_name).get_id(),
            _parent=parent_entity.get_id(),
```

```

        _name=ctx.IDENTIFIER().getText(),
        _longname=parent_file_path,
        _contents=ctx.getText(),
    )

```

Modify deref Listener:

This is the main part of this section's code. Search scope checks current until it finds a type name that is in the type names list; and if it finds a class or method it returns current. In the following two methods we check whether the scope is a class or a method. Therefore we add the entity to our table setting the attributes kind, parent, name, long name using the methods explained before. At last we also add the id and content and return the object. The enter expressions 6 and 21 are for "=" and "+, -, *, ..." expression which can modify a variable. Last but not least we have the main function which after being called from the enter expressions, checks whether there is an opening for a scope and uses search scope to declare that. After that we check if the scope is a class or a method using those two functions explained earlier; and set the correct name and entity kind for each one of them and pass it on to add reference for the final result to be added to our database.

```

class ModifyListener(JavaParserLabeledListener):
    def __init__(self, files, file_name):
        self.files = files
        self.file_name = file_name

    @staticmethod
    def search_scope(ctx, type_names):
        # Traverse bottom up until reaching a class or method
        current = ctx.parentCtx
        while current is not None:
            type_name = type(current).__name__
            if type_name in type_names:
                return current
            current = current.parentCtx
        return None

```

```

def make_scope_class(self, ctx, file_name):
    prefixes = get_prefixes(ctx, "ClassDeclarationContext")
    kind_name = get_kind_name(prefixes, kind="Class")
    kind_id = KindModel.get_or_none(_name=kind_name).get_id()
    name = ctx.IDENTIFIER().getText()
    parent_entity, parent_file_path = get_parent(file_name, self.files)
    content = ctx.getText()

    obj = EntityModel.get_or_none(
        _kind=KindModel.get_or_none(_name=kind_name).get_id(),
        _parent=parent_entity.get_id(),
        _name=name,
        _longname=parent_file_path,
    )
    return {
        "id": obj.get_id(),
        "kind_id": kind_id,
        "parent_id": parent_entity.get_id(),
        "name": name,
        "longname": parent_file_path,
        "content": content
    }

def make_scope_method(self, ctx, file_name):
    prefixes = get_prefixes(ctx, "MethodDeclarationContext")
    kind_name = get_kind_name(prefixes, kind="Method")
    kind_id = KindModel.get_or_none(_name=kind_name).get_id()
    name = ctx.IDENTIFIER().getText()
    content = ctx.getText()
    parent_ctx = self.search_scope(ctx, ["ClassDeclarationContext"])
    parent_entity = self.make_scope_class(parent_ctx, file_name)

    obj, _ = EntityModel.get_or_create(
        _kind=kind_id,
        _parent=parent_entity['id'],
        _name=name,
        _longname=f"{parent_entity['name']}.{name}",
        _contents=content,
    )

```

```

    )
    return {
        "id": obj.get_id(),
        "kind_id": kind_id,
        "parent_id": parent_entity['id'],
        "name": name,
        "longname": f"{{parent_entity['name']}}.{{name}}",
        "content": content
    }

def enterExpression6(self, ctx: JavaParserLabeled.Expression6Context):
    self.modify_deref_partial(ctx)

def enterExpression21(self, ctx: JavaParserLabeled.Expression21Context):
    self.modify_deref_partial(ctx)

def modify_deref_partial(self, ctx):
    lhs_text = ctx.children[0].getText()
    scope = None
    if "[" in lhs_text and "]" in lhs_text:
        var_name = ctx.children[0].children[0].getText()
        var_entity = add_var_entity(var_name)

        line, col = str(ctx.start).split(",")[3][: -1].split(':')

        file_entity, _ = get_parent(self.file_name, self.files)
        file_id = file_entity.get_id()

        ref_dict = {'line': line, 'column': col, 'file_id': file_id, 'text': ctx.getText()}

        scope_ctx = self.search_scope(ctx, ["ClassDeclarationContext",
"MethodDeclarationContext"])
        if type(scope_ctx).__name__ == "ClassDeclarationContext":
            scope = self.make_scope_class(scope_ctx, self.file_name)
        elif type(scope_ctx).__name__ == "MethodDeclarationContext":
            scope = self.make_scope_method(scope_ctx, self.file_name)

        add_references(scope, var_entity, ref_dict)

```

Add variable entity:

Add each variable entity to the database table of entities.

```
def add_var_entity(var_name):
    obj, _ = EntityModel.get_or_create(
        _kind=KindModel.get_or_none(_name="Java Unresolved Variable").get_id(),
        _name=var_name,
        _longname=var_name,
    )
    return {
        "id": obj.get_id(),
        "kind_id": KindModel.get_or_none(_name="Java Unresolved Variable").get_id(),
        "name": var_name,
        "longname": var_name,
    }
```

Add references:

Just as we explained in the last section, this method adds the final objects to our reference table in the database. The kind id is set to either “Java Modify Deref Partial” or “Java Modifyby Deref Partial”. The file, line and column in which the modifying has happened are all set in their attributes. We have also added the entity which does the action and the scope the action has taken place on.

```
def add_references(scope, ent, ref_dict):
    ref, _ = ReferenceModel.get_or_create(
        _kind=KindModel.get_or_none(_name="Java Modify Deref Partial").get_id(),
        _file=ref_dict['file_id'],
        _line=ref_dict['line'],
        _column=ref_dict['column'],
        _ent=ent['id'],
        _scope=scope['id'],
    )
    inverse_ref, _ = ReferenceModel.get_or_create(
        _kind=KindModel.get_or_none(_name="Java Modifyby Deref Partial").get_id(),
        _file=ref_dict['file_id'],
        _line=ref_dict['line'],
```

```

        _column=ref_dict['column'],
        _ent=scope['id'],
        _scope=ent['id'],
    )

```

Main:

After creating an instance of the project and initializing the database and setting the java files; we get then parse tree of each file and traverse through it. We make our listeners and walk through them, as explained before the expressions and scopes of the file are checked for a modified variable in the scope mentioned. After detecting and setting the attributes of each modify deref partial that has happened, we have the final results as a table.

def main():

```

    info = get_project_info(PRJ_INDEX, REF_NAME)
    p = Project(info['DB_PATH'], info['PROJECT_PATH'], info['PROJECT_NAME'])
    p.init_db()
    p.get_java_files()

```

for file_name, file_path in p.files:

```

    tree = get_parse_tree(file_path)
    walker = ParseTreeWalker()

```

```

    class_listener = ClassListener(p.files, file_name)
    listener = ModifyListener(p.files, file_name)

```

```

    walker.walk(class_listener, tree)
    walker.walk(listener, tree)

```

Tests and results

There were several benchmark projects tested and compared, so we will share the final tables of a few results for some of the projects. (The rest can be found on our source files).

For testing our benchmark projects we added a g10 test file:

```
import understand as und
import os

PRJ_INDEX = 0
REF_NAME = "origin"

def get_project_info(index, ref_name):
    project_names = [
        'calculator_app',
        'JSON',
        'testing_legacy_code',
        'jhotdraw-develop',
        'xerces2j',
        'jvlt-1.3.2',
        'jfreechart',
        'ganttproject',
        '105_freemind',
    ]
    project_name = project_names[index]
    db_path = f"../databases/{ref_name}/{project_name}"
    if ref_name == "origin":
        db_path = db_path + ".udb"
    else:
        db_path = db_path + ".oudb"
    project_path = f"../benchmarks/{project_name}"

    db_path = os.path.abspath(db_path)
    project_path = os.path.abspath(project_path)

    return {
        'PROJECT_NAME': project_name,
        'DB_PATH': db_path,
```



```

    'PROJECT_PATH': project_path,
}

def test_understand_kinds():
    info = get_project_info(PRJ_INDEX, REF_NAME)
    db = und.open(info['DB_PATH'])
    for ent in db.ents():
        for ref in ent.refs("Import"):
            print(f'1. ref name: {ref.kindname()}')
            print(f'2. ref scope: {ref.scope().longname()} || kind: {ref.scope().kind()}')
            print(f'3. ref ent: {ref.ent().longname()} || kind: {ref.ent().kind()}')
            print(f'4. file location: {ref.file().longname()} || line: {ref.line()}')
            print("-" * 25)

if __name__ == '__main__':
    test_understand_kinds()

```

Then we compared the results of our database to Understand's results.

The resulted tables are as follows:

Import(calculator app example)

```

1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\calculator_app\src\com\calculator\app\display\println.java || kind: Java File
3. ref ent: com.calculator.app.display.print_success || kind: Java Class Type Public Member
4. file location: println.java || line: 3
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\calculator_app\src\com\calculator\app\display\println.java || kind: Java File
3. ref ent: com.calculator.app.display.print_fail || kind: Java Class Type Public Member
4. file location: println.java || line: 4
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\calculator_app\src\com\calculator\app\display\println.java || kind: Java File
3. ref ent: java.lang.System || kind: Java Unknown Class Type Member
4. file location: println.java || line: 6
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\calculator_app\src\com\calculator\app\display\print_fail.java || kind: Java File
3. ref ent: java.lang.System || kind: Java Unknown Class Type Member
4. file location: print_fail.java || line: 3
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\calculator_app\src\com\calculator\app\display\print_success.java || kind: Java File
3. ref ent: java.lang.System || kind: Java Unknown Class Type Member
4. file location: print_success.java || line: 3
-----

```

Figure 1.1 CMD results

Compiler Project

The screenshot shows the DB Browser for SQLite interface. The main table displayed is the Database entity table, which lists 16 rows of data. The table has columns: _id, kind_id, _parent, _name, _longname, _value, _type, and _contents. The data is as follows:

_id	kind_id	_parent	_name	_longname	_value	_type	_contents
1	1	1	println.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.display;...
2	2	1	print_fail.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.display;...
3	3	1	print_success.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.display;...
4	4	1	Main.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.init;...
5	5	1	basic_operation.j...	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.method;...
6	6	1	fibonacci.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.method;...
7	7	1	integral.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.method;...
8	8	1	printLog.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package com.calculator.app.method;...
9	9	98	3 print_success	com.calculator.app.display.print_success	NULL	NULL	classprint_success(publicstaticvoidpr...
10	10	98	2 print_fail	com.calculator.app.display.print_fail	NULL	NULL	classprint_fail(publicvoidprint_fail_m...
11	11	84	NULL System	java.lang.System	NULL	NULL	NULL
12	12	98	7 integral	com.calculator.app.method.integral	NULL	NULL	classintegralextendsbasic_operation(...
13	13	98	6 fibonacci	com.calculator.app.method.fibonacci	NULL	NULL	classfibonacciextendsbasic_operation...
14	14	98	1 println	com.calculator.app.display.println	NULL	NULL	classprintln(publicvoidprint(Stringtex...
15	15	98	4 Main	com.calculator.app.init.Main	NULL	NULL	classMain(publicstaticvoidmain(Strin...
16	16	98	8 printLog	com.calculator.app.method.printLog	NULL	NULL	classprintLogextendsprintln(@Overr...

Figure 1.2 Database entity table

The screenshot shows the DB Browser for SQLite interface. The main table displayed is the Database reference table, which lists 23 rows of data. The table has columns: _id, kind_id, _file_id, _line, _column, _ent_id, and _scope_id. The data is as follows:

_id	kind_id	_file_id	_line	_column	_ent_id	_scope_id
1	1	206	1	3	0	9
2	2	207	1	3	0	1
3	3	206	1	4	0	10
4	4	207	1	4	0	1
5	5	206	1	6	0	11
6	6	207	1	6	0	1
7	7	206	2	3	0	11
8	8	207	2	3	0	2
9	9	206	3	3	0	11
10	10	207	3	3	0	3
11	11	206	4	3	0	12
12	12	207	4	3	0	4
13	13	206	4	4	0	13
14	14	207	4	4	0	4
15	15	206	4	5	0	9
16	16	207	4	5	0	4
17	17	206	4	6	0	14
18	18	207	4	6	0	4
19	19	206	5	3	0	15
20	20	207	5	3	0	5
21	21	206	7	3	0	16
22	22	207	7	3	0	7
23	23	206	8	3	0	14

Figure 1.3 Database reference table

```
D:\Documents\University\Semesters\Term 8\Compiler\Project P1\OpenUnderstand\openunderstand>python gl0_tests.py
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\method\integral.java || kind: File
3. ref ent: com.calculator.app.method.printLog || kind: Public Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\method\integral.java || line: 3
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\println.java || kind: File
3. ref ent: com.calculator.app.display.print_success || kind: Public Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\println.java || line: 3
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\println.java || kind: File
3. ref ent: com.calculator.app.display.print_fail || kind: Public Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\println.java || line: 4
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\println.java || kind: File
3. ref ent: java.lang.System || kind: Unknown Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\println.java || line: 6
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || kind: File
3. ref ent: com.calculator.app.method.integral || kind: Public Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || line: 3
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || kind: File
3. ref ent: com.calculator.app.method.fibonacci || kind: Public Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || line: 4
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || kind: File
3. ref ent: com.calculator.app.display.print_success || kind: Public Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || line: 5
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || kind: File
3. ref ent: com.calculator.app.display.println || kind: Public Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\init\Main.java || line: 6
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\print_success.java || kind: File
3. ref ent: java.lang.System || kind: Unknown Class
4. file location: C:\Users\alise\Desktop\calculator_app\src\com\calculator\app\display\print_success.java || line: 3
```

Figure 1.4 Understand results

Import(xerces2j example)

```

1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\xerces2j\src\org\apache\html\dom\HTMLDocumentImpl.java || kind: Java File
3. ref ent: org.w3c.dom.html.HTMLTitleElement || kind: Java Unknown Class Type Member
4. file location: HTMLDocumentImpl.java || line: 40
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\xerces2j\src\org\apache\html\dom\HTMLDOMImplementationImpl.java || kind: Java File
3. ref ent: org.apache.xerces.dom.DOMImplementationImpl || kind: Java Class Type Public Member
4. file location: HTMLDOMImplementationImpl.java || line: 20
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\xerces2j\src\org\apache\html\dom\HTMLDOMImplementationImpl.java || kind: Java File
3. ref ent: org.w3c.dom.DOMException || kind: Java Unknown Class Type Member
4. file location: HTMLDOMImplementationImpl.java || line: 21
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\xerces2j\src\org\apache\html\dom\HTMLDOMImplementationImpl.java || kind: Java File
3. ref ent: org.w3c.dom.html.HTMLDOMImplementation || kind: Java Interface Type Public
4. file location: HTMLDOMImplementationImpl.java || line: 22
-----
1. ref name: Java Import
2. ref scope: D:\Documents\University\Semesters\Term 8\Compiler\Project P1\benchmarks\xerces2j\src\org\apache\html\dom\HTMLDOMImplementationImpl.java || kind: Java File
3. ref ent: org.w3c.dom.html.HTMLDocument || kind: Java Unknown Class Type Member
4. file location: HTMLDOMImplementationImpl.java || line: 23
-----

```

Figure 2.1 CMD results

DB Browser for SQLite - D:\Documents\University\Semesters\Term 8\Compiler\Project P1\database\import\xerces2j.oudb

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

Table: entitymodel

_id	kind	_parent	_name	_longname	_value	_type	_contents
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	NULL	HTMLAnchorElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	/*
2	1	NULL	HTMLAppletElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	* Licensed to the Apache Software Foundation
3	1	NULL	HTMLAreaElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	(ASF) under one or more
4	1	NULL	HTMLBaseElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	* contributor license agreements. See the
5	1	NULL	HTMLBaseFontElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	NOTICE file distributed with
6	1	NULL	HTMLBodyElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	* this work for additional information regarding
7	1	NULL	HTMLBRElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	copyright ownership.
8	1	NULL	HTMLBuilder.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
9	1	NULL	HTMLButtonElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
10	1	NULL	HTMLCollectionImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
11	1	NULL	HTMLDirectoryElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
12	1	NULL	HTMLDivElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
13	1	NULL	HTMLDListElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
14	1	NULL	HTMLDocumentImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
15	1	NULL	HTMLDOMImplementationImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
16	1	NULL	HTMLElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
17	1	NULL	HTMLFieldSetElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
18	1	NULL	HTMLFontElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
19	1	NULL	HTMLFormControl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
20	1	NULL	HTMLFormElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
21	1	NULL	HTMLFrameElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
22	1	NULL	HTMLFrameSetElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	
23	1	NULL	HTMLHeadElementImpl.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	

1 - 24 of 1331

Go to: 1

Edit Database Cell

Mode: Text

```

1 /*
2 * Licensed to the Apache Software Foundation
3 (ASF) under one or more
4 * contributor license agreements. See the
5 NOTICE file distributed with
6 * this work for additional information regarding
7 copyright ownership.

```

Type of data currently in cell: Text / Numeric
4708 character(s)

Remote

Identity Select an identity to connect

DBHub.io Local Current Database

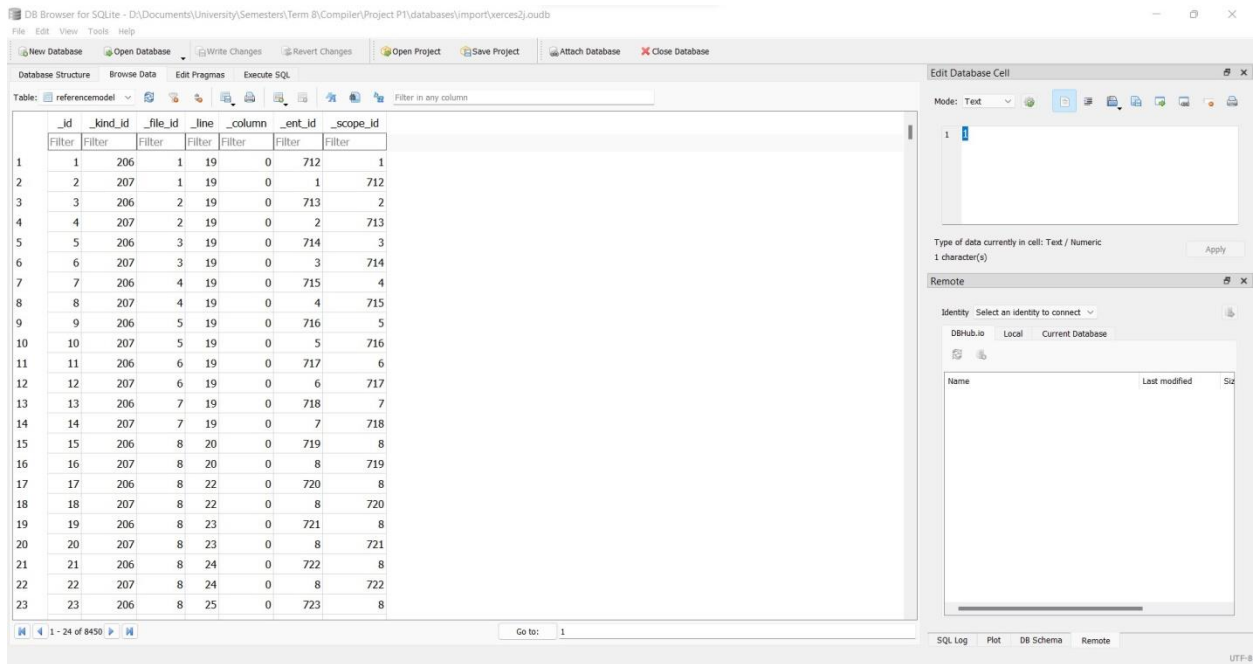
Name	Last modified	Size
------	---------------	------

SQL Log Plot DB Schema Remote

UTF-8

Figure 2.2 Database entity table

Compiler Project



The screenshot shows the DB Browser for SQLite interface. The main window displays a table named 'referencemodel' with 7 columns: _id, _kind_id, _file_id, _line, _column, _ent_id, and _scope_id. The table contains 23 rows of data. On the right, there are two panels: 'Edit Database Cell' and 'Remote'. The 'Edit Database Cell' panel shows a text input field with the value '1'. The 'Remote' panel shows a list of remote databases with columns for Name, Last modified, and Size.

_id	_kind_id	_file_id	_line	_column	_ent_id	_scope_id
1	206	1	19	0	712	1
2	207	1	19	0	1	712
3	206	2	19	0	713	2
4	207	2	19	0	2	713
5	206	3	19	0	714	3
6	207	3	19	0	3	714
7	206	4	19	0	715	4
8	207	4	19	0	4	715
9	206	5	19	0	716	5
10	207	5	19	0	5	716
11	206	6	19	0	717	6
12	207	6	19	0	6	717
13	206	7	19	0	718	7
14	207	7	19	0	7	718
15	206	8	20	0	719	8
16	207	8	20	0	8	719
17	206	8	22	0	720	8
18	207	8	22	0	8	720
19	206	8	23	0	721	8
20	207	8	23	0	8	721
21	206	8	24	0	722	8
22	207	8	24	0	8	722
23	206	8	25	0	723	8

Figure 2.3 Database reference table

```
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\SecuritySupport.java || kind: File
3. ref ent: java.security.PrivilegedAction || kind: Unknown Class
4. file location: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\SecuritySupport.java || line: 25
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\SecuritySupport.java || kind: File
3. ref ent: java.security.PrivilegedActionException || kind: Unknown Class
4. file location: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\SecuritySupport.java || line: 26
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\SecuritySupport.java || kind: File
3. ref ent: java.security.PrivilegedActionException || kind: Unknown Class
4. file location: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\SecuritySupport.java || line: 27
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || kind: File
3. ref ent: org.apache.xerces.dom.events.EventImpl || kind: Public Class
4. file location: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || line: 31
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || kind: File
3. ref ent: org.apache.xerces.dom.events.MouseEventImpl || kind: Public Class
4. file location: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || line: 32
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || kind: File
3. ref ent: org.apache.xerces.dom.events.MutationEventImpl || kind: Public Class
4. file location: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || line: 33
-----
1. ref name: Import
2. ref scope: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || kind: File
3. ref ent: org.apache.xerces.dom.events.UIEventImpl || kind: Public Class
4. file location: C:\Users\alise\Desktop\benchmarks\xerces2j\src\org\apache\xerces\dom\DocumentImpl.java || line: 34
```

Figure 2.4 Understand results

Modify deref(jhotdraw-develop example)

```

ctype['-']=CT_DIGIT
1. ref name: Java Modify Deref Partial
2. ref scope: StreamPosTokenizer.parseNumbers || kind: Java Method Default Member
3. ref ent: ctype || kind: Java Variable
4. file location: StreamPosTokenizer.java || line: 349
-----
ctype['+']=CT_DIGIT
1. ref name: Java Modify Deref Partial
2. ref scope: StreamPosTokenizer.parsePlusAsNumber || kind: Java Method Default Member
3. ref ent: ctype || kind: Java Variable
4. file location: StreamPosTokenizer.java || line: 354
-----
buf[i++]=(char)c
1. ref name: Java Modify Deref Partial
2. ref scope: StreamPosTokenizer.nextToken || kind: Java Method Default Member
3. ref ent: buf || kind: Java Variable
4. file location: StreamPosTokenizer.java || line: 727
-----
buf[i++]=(char)c
1. ref name: Java Modify Deref Partial
2. ref scope: StreamPosTokenizer.nextToken || kind: Java Method Default Member
3. ref ent: buf || kind: Java Variable
4. file location: StreamPosTokenizer.java || line: 803

```

Figure 3.1 CMD results

Compiler Project

The screenshot shows the DB Browser for SQLite interface. The main table displayed is 'entitymodel'. The table has columns: _id, _kind_id, _parent_id, _name, _longname, _value, _type, and _contents. The data is filtered to show rows 888 to 910. The right sidebar shows the 'Edit Database Cell' dialog with the text 'Close handle' and the 'Remote' tab selected.

_id	_kind_id	_parent_id	_name	_longname	_value	_type	_contents
888	888	33	883 drawParagraph	TextAreaFigure.drawParagraph	NULL	NULL	Rectangle2D.Double.drawParagraph(...)
889	889	33	883 getPreferredSize	TextAreaFigure.getPreferredSize	NULL	NULL	Dimension2D.Double.getPreferredSize...
890	890	98	206 TextFigure	D:...	NULL	NULL	classTextFigureextendsAbstractAttri...
891	891	98	208 TriangleFigure	D:...	NULL	NULL	classTriangleFigureextendsAbstractA...
892	892	79	209 AbstractConnectionHandle	D:...	NULL	NULL	classAbstractConnectionHandleexten...
893	893	79	210 AbstractHandle	D:...	NULL	NULL	classAbstractHandleimplementsHand...
894	894	79	211 AbstractRotateHandle	D:...	NULL	NULL	classAbstractRotateHandleextendsA...
895	895	98	212 BezierControlPointHandle	D:...	NULL	NULL	classBezierControlPointHandleexten...
896	896	152	newValue.x	newValue.x	NULL	NULL	
897	897	33	895 trackEnd	BezierControlPointHandle.trackEnd	NULL	NULL	voidtrackEnd(Pointanchor,Pointlead,...)
898	898	152	newValue.y	newValue.y	NULL	NULL	
899	899	98	213 BezierNodeHandle	D:...	NULL	NULL	classBezierNodeHandleextendsAbstr...
900	900	98	214 BezierOutlineHandle	D:...	NULL	NULL	classBezierOutlineHandleextendsAbs...
901	901	98	215 BezierScaleHandle	D:...	NULL	NULL	classBezierScaleHandleextendsAbstr...
902	902	98	216 BoundsOutlineHandle	D:...	NULL	NULL	classBoundsOutlineHandleextendsAb...
903	903	98	217 CloseHandle	D:...	NULL	NULL	classCloseHandleextendsLocatorHan...
904	904	98	218 ConnectionEndHandle	D:...	NULL	NULL	classConnectionEndHandleextendsAb...
905	905	98	219 ConnectionStartHandle	D:...	NULL	NULL	classConnectionStartHandleextendsA...
906	906	98	220 ConnectorHandle	D:...	NULL	NULL	classConnectorHandleextendsAbstrac...
907	907	98	221 DragHandle	D:...	NULL	NULL	classDragHandleextendsAbstractHan...
908	908	98	222 FontSizeHandle	D:...	NULL	NULL	classFontSizeHandleextendsLocatorH...
909	909	98	224 HandleAttributeKeys	D:...	NULL	NULL	classHandleAttributeKeys(privatestat...
910	910	79	225 LocatorHandle	D:...	NULL	NULL	classLocatorHandleextendsAbstractH...

Figure 3.2 Database entity table

The screenshot shows the DB Browser for SQLite interface. The main table displayed is 'referencemodel'. The table has columns: _id, _kind_id, _file_id, _line, _column, _ent_id, and _scope_id. The data is filtered to show rows 1 to 23. The right sidebar shows the 'Edit Database Cell' dialog with the text '1' and the 'Remote' tab selected.

_id	_kind_id	_file_id	_line	_column	_ent_id	_scope_id
1	1	210	71	227	16	732
2	2	211	71	227	16	733
3	3	210	71	234	20	732
4	4	211	71	234	20	734
5	5	210	71	237	20	732
6	6	211	71	237	20	734
7	7	210	71	242	8	732
8	8	211	71	242	8	735
9	9	210	83	953	12	747
10	10	211	83	953	12	748
11	11	210	85	438	16	751
12	12	211	85	438	16	752
13	13	210	89	1302	12	747
14	14	211	89	1302	12	758
15	15	210	187	432	12	865
16	16	211	187	432	12	866
17	17	210	187	433	12	867
18	18	211	187	433	12	866
19	19	210	205	103	20	884
20	20	211	205	103	20	885
21	21	210	205	111	28	886
22	22	211	205	111	28	885
23	23	210	205	158	16	887

Figure 3.3 Database reference table

Modify deref(JSON example)

```

names[i]=fields[i].getName()
1. ref name: Java Modify Deref Partial
2. ref scope: JSONObject.getNames || kind: Java Method Default Member
3. ref ent: names || kind: Java Variable
4. file location: JSONObject.java || line: 844
-----
chars[pos]=this.next()
1. ref name: Java Modify Deref Partial
2. ref scope: JSONTokenizer.next || kind: Java Method Default Member
3. ref ent: chars || kind: Java Variable
4. file location: JSONTokenizer.java || line: 275
-----
this.stack[this.top]=jo
1. ref name: Java Modify Deref Partial
2. ref scope: JSONWriter.push || kind: Java Method Default Member
3. ref ent: this.stack || kind: Java Variable
4. file location: JSONWriter.java || line: 295
-----
circle[i]=c
1. ref name: Java Modify Deref Partial
2. ref scope: XMLTokenizer.skipPast || kind: Java Method Default Member
3. ref ent: circle || kind: Java Variable
4. file location: XMLTokenizer.java || line: 371
-----
circle[offset]=c
1. ref name: Java Modify Deref Partial
2. ref scope: XMLTokenizer.skipPast || kind: Java Method Default Member
3. ref ent: circle || kind: Java Variable
4. file location: XMLTokenizer.java || line: 409

```

Figure 4.1 CMD results

Compiler Project

The screenshot shows the DB Browser for SQLite interface. The main table displayed is the 'entitymodel' table, which contains 44 rows of data. The table has columns: _id, _kind_id, _parent_id, _name, _longname, _value, _type, and _contents. The data represents various Java classes and methods from the Compiler Project.

_id	_kind_id	_parent_id	_name	_longname	_value	_type	_contents
22	22	1	XMLXsiTypeConverter.java	D:\Documents\University\Semesters\Term ...	NULL	NULL	package org.json;...
23	23	98	1 CDL	D:\Documents\University\Semesters\Term ...	NULL	NULL	classCDL(privateStringgetValue(JSO...
24	24	98	2 Cookie	D:\Documents\University\Semesters\Term ...	NULL	NULL	classCookie(publicstaticStringescape...
25	25	98	3 CookieList	D:\Documents\University\Semesters\Term ...	NULL	NULL	classCookieList(publicstaticJSONObje...
26	26	98	4 HTTP	D:\Documents\University\Semesters\Term ...	NULL	NULL	classHTTP(publicstaticfinalStringCRL...
27	27	98	5 HTTPTokener	D:\Documents\University\Semesters\Term ...	NULL	NULL	classHTTPTokenerextendsJSONToke...
28	28	98	6 JSONArray	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONArrayimplementsIterable<...
29	29	98	7 JSONException	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONExceptionextendsRuntimeE...
30	30	98	8 JSONML	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONML(privatestaticObjectpars...
31	31	98	9 JSONObject	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONObject(privatestaticfinalda...
32	32	95	9 Null	D:\Documents\University\Semesters\Term ...	NULL	NULL	classNull{@OverrideprotectedfinalOb...
33	33	152	names	names	NULL	NULL	
34	34	33	31 getNames	JSONObject.getNames	NULL	NULL	String[]getNames(Objectobject)...
35	35	98	10 JSONPointer	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONPointer(privatestaticfinalSt...
36	36	95	10 Builder	D:\Documents\University\Semesters\Term ...	NULL	NULL	classBuilder(privatefinalList<String>r...
37	37	98	11 JSONPointerException	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONPointerExceptionextendsJS...
38	38	98	15 JSONStringer	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONStringerextendsJSONWriter...
39	39	98	16 JSONTokener	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONTokener(privatelongcharac...
40	40	152	chars	chars	NULL	NULL	
41	41	33	39 next	JSONTokener.next	NULL	NULL	Stringnext(intn)throwsJSONExceptio...
42	42	98	17 JSONWriter	D:\Documents\University\Semesters\Term ...	NULL	NULL	classJSONWriter(privatestaticfinalInt...
43	43	152	this.stack	this.stack	NULL	NULL	
44	44	33	42 push	JSONWriter.push	NULL	NULL	voidpush(JSONObjectjo)throwsJSON...

Figure 4.2 Database entity table

The screenshot shows the DB Browser for SQLite interface. The main table displayed is the 'referencemodel' table, which contains 10 rows of data. The table has columns: _id, _kind_id, _file_id, _line, _column, _ent_id, and _scope_id. The data represents references between different parts of the project.

_id	_kind_id	_file_id	_line	_column	_ent_id	_scope_id
1	1	210	9	844	12	33
2	2	211	9	844	12	34
3	3	210	16	275	12	40
4	4	211	16	275	12	41
5	5	210	17	295	8	43
6	6	211	17	295	8	44
7	7	210	21	371	12	49
8	8	211	21	371	12	50
9	9	210	21	409	12	49
10	10	211	21	409	12	50

Figure 4.3 Database reference table

Procedure and Challenges

As we all know in every project there are many challenges and problems waiting for us. The first step to every project is always one of the hardest ones since it is like entering a whole new world. As for this project our team tried to read and search about the purpose of this project, so the first step was to find out what Understand is and what it does as a tool.

After searching different sources and reading some parts of this tool's user manual; we figured out that going through your codes for a simple purpose or error is very time and energy consuming, while this tool simply allows you to see most of the aspects your code's structure has.

After figuring out the necessity of this project and attending to the classes with additional notes about the project structure, we had a 5 days period to learn more about the structure of the code and our tasks, by reading the documentation of this project. Then we installed and tried working with different tools needed to complete this project such as DB browser for SQLite and understand itself.

After all the setup was done we tried to study the codes in the GitHub repository and detect the main problems and issues. The next step was how to divide the work between us, although most of the time the team worked together but some of the parts were done individually. The challenging part was how to handle different entity kinds and add all the necessary data to our database tables correctly. Apart from that issue considering all of the possible options was confusing, hard and time consuming. But we tried to get help from the documentation and the type of data that we could see in the parse tree and Understand's supported types. The implementation was done but we still needed to test it, we tried running one of the test cases and checking the tables, which seemed like a success. We also compared it to the results that understand would give us. But as we tried different benchmarks the execution speed was slow and ended up troubling us.

The last part was to write a documentation that indicates all the work and effort done during this time. We tried to write this documentation as simple and clear as possible, that is to explain every part of our code. But because of the connections the codes have it was hard to maintain a fine flow for the reader's mind and we hope that we succussed doing so.

Conclusions and Recommendations

After finishing this project we think Understand is a useful tool and having an open source Api can help a lot to customize and analyze your code. Now that we can spot out each import we can take control of our sources, inheritance and structure of our code, therefore we can prevent spaghetti code from happening and get one step closer to having a clean code. It goes the same for modify, detecting where our entities and variables are changing can help a lot with debugging and to prevent with changes in wrong scopes for our variables.

For future work this project can be developed to much more completed version of itself for example we can add more entity detection and analyze the code more so that each reference has more attributes. After doing so all of the features have to go through more filters of testing, since the runtime isn't in the favor of many test the codes should be optimized many times, each time getting better and more efficient.

Additional Sources

<https://www.scitools.com/>

<https://documentation.scitools.com/pdf/understand.pdf>

<https://m-zakeri.github.io/OpenUnderstand/>

[https://en.wikipedia.org/wiki/Understand_\(software\)](https://en.wikipedia.org/wiki/Understand_(software))