Compiler project document

Group 4

Group members:

Shaghayegh Mobasher 97522004

Setayesh Kouloubandi 9752202

Parisa Alaie 97522175

Contents

1) Introduction

In this project, Group members have tried to implement open source Understand Python API to analyze source code. In fact, this was the development of an open source implementation of Understand Python API.

We first worked on implementing the API for Java applications using Python programming languages and compiler tools such as Antlr4. And examines the structures used by Understand to analyze source code.

For the G4 group, they had to analyze to find a subset of the list of reference types for implement Couple_implementbyCouple by, Create Create by and declare declare in with their respective entities.

Items that are used in the project and should be introduced here:

Entity: Anything in code that Understand gets information about: for example, a file, a class, a variable, a function, and so on. It used to store Java entities in the project. This table is populated by ANTLR Listener during static analysis of the program. It has a unique set of features that can be searched by the API

Reference: The specific place where an entity appears in code. A reference is always defined as the relationship between two entities. It has both related entities as well as the file, line and column in which the reference is placed and its type of authority. It has a unique set of features that can be searched by the API

Project: To store some basic project information under analysis such as project name, programming languages and so on. This table is filled in automatically.

Kind: To store both types of entities and references, which can be separated by auto-filling the database and the necessary numbers and related to different sections.

The peewee and SQLite3 libraries are used for the project database, and most of the data collected includes Entity and Reference.

The division of labor for this project has been such that each person has to do a necessary part of three parts:

1. Parisa Alaie section implement Couple _ implement by Couple by
2. Setayesh Koloubandi section Create by section
3. Shaghayegh Mobasher section declare declare in

In the following order We will pay for:

- The proposed method that includes
    - The main code, how to find
    - How to find and Set entity, parent
    - Implement declare declarein
    - How to use declare declare in Listener
    - Implement Couple _ implement by Couple by
    - How to use implement Couple _ implement by Couple by in Listener
    - Implementation Create Create by
    - How to use Create Create by in Listener
- Assessment
    - Which we compare the obtained answers with the answers that our understanding gives
- Conclusions and future work
    - Show the overall result of the program is written.

2) Methodology

General codes for all parts (main.py)

In this part, first, a basic database is constructed. A loop is made for every Java file of the project in the given folder (path) of all references of the project.

For each file in the loop, the entity of the file is returned first, using (getFileEntity in Project class), and then the “listener’s reference” is created. The listener and the file’s path for creating FileStream are given to the ParseAndWalk method.

This method will walk the tree and the listener is filled with the list of the references’ properties. Then the special method for that reference is called to fill the database.

Error handling is done here so that if there is a problem in a particular file the other files will be processed.

Finding a class/interface’s database fields (class_properties.py)

```
def enterClassDeclaration(self, ctx:JavaParserLabeled.ClassDeclarationContext):
  if self.class_properties:  # already found the class
     return
  if self.class_longname[-1] == ctx.IDENTIFIER().getText():
     if self.checkParents(ctx):
        # this is the exact class we wanted.
        self.class_properties = {}
        self.class_properties["name"] = self.class_longname[-1]
        self.class_properties["longname"] = ".".join(self.class_longname)

        if len(self.class_longname) == 1:
           self.class_properties["parent"] = None
        else:
           self.class_properties["parent"] = self.class_longname[-2]
        self.class_properties["modifiers"] =
ClassPropertiesListener.findClassOrInterfaceModifiers(ctx)
        self.class_properties["contents"] = ctx.getText()
```

```
class InterfacePropertiesListener(JavaParserLabeledListener):
  interface_longname = []
  interface_properties = None

  def checkParents(self, c):
    return set(ClassPropertiesListener.findParents(c)) &
set(list(reversed(self.interface_longname)))

  def enterInterfaceDeclaration(self,
ctx:JavaParserLabeled.InterfaceDeclarationContext):
    if self.interface_properties:  # already found the class
      return
    if self.interface_longname[-1] == ctx.IDENTIFIER().getText():
      if self.checkParents(ctx):
        # this is the exact class we wanted.
        self.interface_properties = {}
        self.interface_properties["name"] = self.interface_longname[-1]
        self.interface_properties["longname"] = ".".join(self.interface_longname)

        if len(self.interface_longname) == 1:
          self.interface_properties["parent"] = None
        else:
          self.interface_properties["parent"] = self.interface_longname[-2]
        self.interface_properties["modifiers"] =
ClassPropertiesListener.findClassOrInterfaceModifiers(ctx)
        self.interface_properties["contents"] = ctx.getText()
```

To find the fields of a class or interface that only provided its long name in the listeners made for the references Create/CreateBy or Implement/ImplementBy, we constructed two listeners ClassPropertiesListener and InterfacePropertiesListener.
These two listeners are almost the same, one made for a class declaration and the other for an interface declaration. This is why only ClassPropertiesListener is discussed here.
To use the listener, the longname of the class should be in the form of a python list starting from the first ancestor's name to the class name itself. The other property in this class is class_properties that will be filled with the class's needed fields in case the class is found.

The only rule considered in the listener is classDeclaration in the method “enterClassDeclaration”:
In this method first we check if the class with this longname has already been found. If that’s the case, there’s no need to continue checking this class declaration.
Otherwise, the name of the class (IDENTIFIER) is compared to the name of the class we look for. Then the parents of this class are compared to the parents mentioned in the longname; this is done in the findParents method. If the class and its ancestor’s name all match this class declaration, we have found the right class.
The fields name, longname and parent in the Entity class can be found from the longname. To find the modifiers of the class (like public or static) we use the static method “findClassOrInterfaceModifiers”. The modifiers can be used to generate the right kind for this class entity. The text of the class declaration is used as the contents of this entity.

Finding the parents of an entity via the “findParents” method

```
@staticmethod
def findParents(c):  # includes the ctx identifier
  parents = []
  current = c
  while current is not None:
    if type(current).__name__ == "ClassDeclarationContext" or
type(current).__name__ == "MethodDeclarationContext"\
          or type(current).__name__ == "EnumDeclarationContext"\
          or type(current).__name__ == "InterfaceDeclarationContext"\
          or type(current).__name__ == "AnnotationTypeDeclarationContext":
        parents.append(current.IDENTIFIER().getText())
    current = current.parentCtx
  return list(reversed(parents))
```

This method loops on the parents’ contexts of a given context. If the context of the parent is for one of the rules “EnumDeclaration”, “ClassDeclaration”, “InterfaceDeclaration” or “AnnotationTypeDeclaration” (which have an IDENTIFIER (name) inside), the IDENTIFIER’s text is considered as the parent’s name and added to the parents list. In the end

we return the reverse of this list to have its most far ancestor at the start of the list.

Finding the modifiers of a class, interface or enum entity via “findClassOrInterfaceModifiers”

```
@staticmethod
def findClassOrInterfaceModifiers(c):
  m = ""
  modifiers=[]
  current = c
  while current is not None:
    if "typeDeclaration" in type(current.parentCtx).__name__:
      m=(current.parentCtx.classOrInterfaceModifier())
      break
    current = current.parentCtx
  for x in m:
    modifiers.append(x.getText())
  return modifiers
```

The type declaration rule has all kinds of modifiers we need to construct the kind of the entity. This is why by iterating over this context’s parent context and finding the first “typeDeclaration”, we can find all the modifiers in its classOrInterfaceModifier rule. These modifiers are then added to a list and returned.

Finding or Creating Entity Models in Class Project (main.py file):

File entity (getFileEntity)

```
def getFileEntity(self, path):
  # kind id: 1
  path = path.replace("/", "\\")
  name = path.split("\\")[-1]
```

```
  file = open(path, mode='r')
  file_ent = EntityModel.get_or_create(_kind=1, _name=name, _longname=path,
_contents=file.read())[0]
  file.close()
  print("file entity",file_ent)
  return file_ent
```

This method gets the name and longname of the entity file using it's address. Also by opening the file the contents will be read and all the fields in addition to "kind id" (which is 1 in database) will be used for getting or creating the entity file.

Package entity (getPackageEntity)

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

The package fields are "kind,name,parent and longname" and also its "contents" which is an empty string.

Package Entity without a Name (getUnnamedPackageEntity)

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

The difference between this method and the above method is in "kind id , name and longname". The latter two is "Unnamed_Package".

Class Entity (getClassEntity)

```
def getClassEntity(self, class_longname, file_address):
  props = p.getClassProperties(class_longname, file_address)
  if not props:  # This class is unknown, unknown class id: 84
    ent = EntityModel.get_or_create(_kind=84, _name=class_longname.split(".")[-1],
                                    _longname=class_longname, _contents="")
  else:
    if len(props["modifiers"]) == 0:
      props["modifiers"].append("default")
    kind = self.findKindWithKeywords("Class", props["modifiers"])
    ent = EntityModel.get_or_create(_kind=kind, _name=props["name"],
                                    _longname=props["longname"],
                                    _parent= props["parent"] if props["parent"] is not None else
file_ent,
                                    _contents=props["contents"])
  return ent[0]
```

To get the entity of a class using file address and “longname” of the class this method is used.

First using “fileaddress” and “longname” the “ClassPropertiesListener” will try to find this class in the file. If it can not find it, it means that the class is not defined in the file and it is necessary for the entity to be an “unknown class”.
Otherwise, the exact class type has to be found using “findKindWithKeywords “. This method gets the entity model (“Class”) and it’s modifiers and returns the best “kind”. Then the class entity will be obtained using “get_or_create”. If it does not have parent in the dictionary it means the parent is the file itself.

Interface Entity (getInterfaceEntity)

```
def getInterfaceEntity(self, interface_longname, file_address): # can't be of unknown
kind!
  props = p.getInterfaceProperties(interface_longname, file_address)
  if not props:
    return None
```

```
  else:
      print ("props[parent] getInterfaceEntity",props["parent"])
      print("koleeeee in -parent in interface", props["parent"] if props["parent"] is not
None else file_ent)
      kind = self.findKindWithKeywords("Interface", props["modifiers"])
      ent = EntityModel.get_or_create(_kind=kind, _name=props["name"],
                                      _longname=props["longname"],
                                      _parent= props["parent"] if props["parent"] is not None else
file_ent,
                                      _contents=props["contents"])
  return ent[0]
```

To get the entity of an interface using file address and “longname” of the interface this method is used. First using “fileaddress” and “longname” the “InterfacePropertiesListener ” will try to find this interface in the file.

If it cannot do this, since we don’t have an unknown interface type, “None” will be returned. That method, which has been called this method, will consider that a class inplace of an interface.

Otherwise, the exact interface type has to be found using “findKindWithKeywords “. This method gets the entity model (“interface”) and it’s modifiers and returns the best “kind”. Then the interface entity will be obtained using “get_or_create”. If it does not have parent in the dictionary it means the parent is the file itself.

Help method findKindWithKeywords (main.py file)

```
def findKindWithKeywords(self, type, modifiers):
  if len(modifiers) == 0:
      modifiers.append("default")
  leastspecific_kind_selected = None
  for kind in KindModel.select().where(KindModel._name.contains(type)):
      if self.checkModifiersInKind(modifiers, kind):
          if not leastspecific_kind_selected \
                  or len(leastspecific_kind_selected._name) > len(kind._name):
              leastspecific_kind_selected = kind
  return leastspecific_kind_selected
```

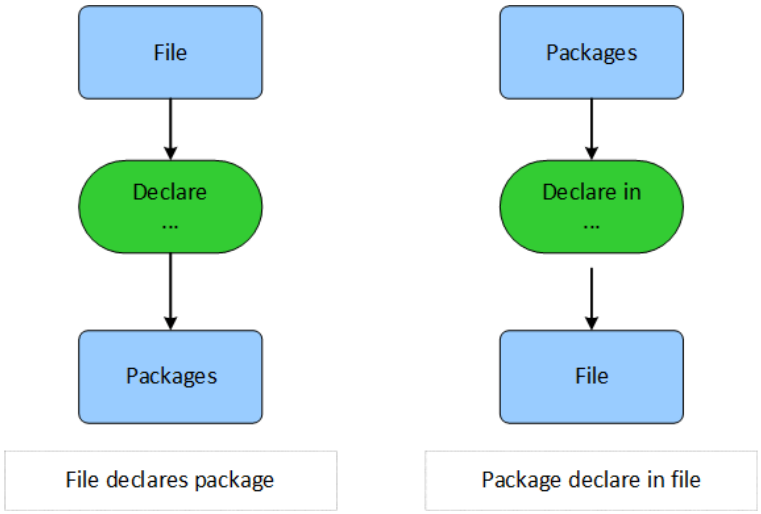
```
def checkModifiersInKind(self, modifiers, kind):
  for modifier in modifiers:
    if modifier.lower() not in kind._name.lower():
      return False
  return True
```

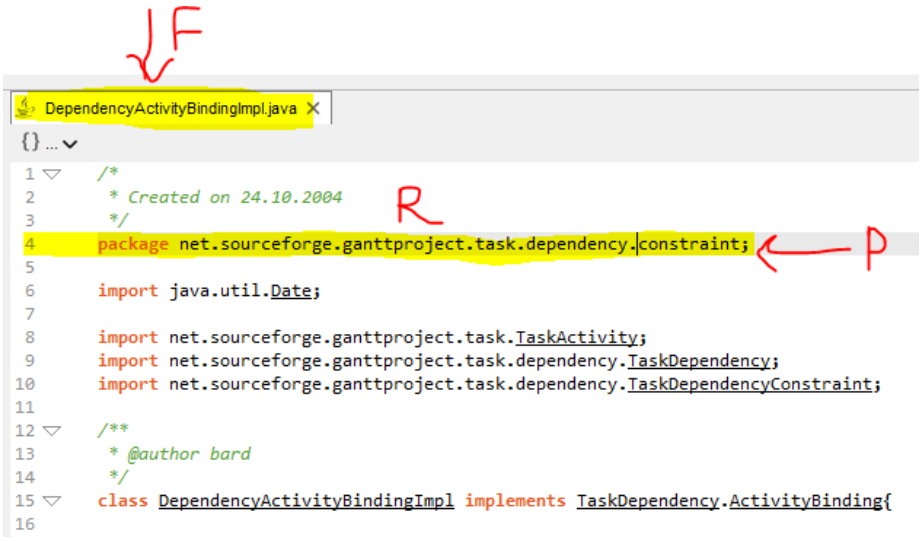
In this method “type” means (“Class, interface or Enum”) and “modifiers” means (“public or static or … “). If “modifiers” is empty a “ default ” is added to it to find it’s default type.

Then, among all “kinds” with “type” in them and those which have “modifiers” in them the shortest “kind” will be returned, which probably do not have the added properties not found in the modifiers. For example, If “modifiers” just have public in them and we are also looking for a class, between “Abstract Class Type Public Member” and “Java Class Type Public Member” it will choose the shortest one which is what we are looking for.

Declare , Declare in references (declare_declarein.py file)

Declare references are made when a package is defined.All files declare one package and if the name of the package is not mentioned in the start of the file that package will add to the entities as an “unnamed package”. Each package also declares the packages that are followed and separated by dot in the name.






```
class DeclareAndDeclareinListener(JavaParserLabeledListener):
    """
    #Todo: Implementing the ANTLR listener pass for Java Call and Java Callby reference kind

    """
    declare = []

    def enterCompilationUnit(self, ctx:JavaParserLabeled.CompilationUnitContext):
        if not ctx.packageDeclaration():  # unnamed package
            self.declare.append({
                "scope": None, "ent": None,
                "line": 1, "col": 0
```

```
        })

    def enterPackageDeclaration(self,
ctx:JavaParserLabeled.PackageDeclarationContext):
        all_declared = ctx.qualifiedName().IDENTIFIER()
        longname = ""
        for i in range(len(all_declared)):
            ent_name = all_declared[i].getText()
            ent_longname = longname + ("." if longname != "" else "") + ent_name
            self.declare.append({
                "scope": all_declared[i-1].getText() if i != 0 else None, "ent": ent_name,
                "scope_longname": longname, "ent_longname": ent_longname,
                "line": all_declared[i].symbol.line, "col": all_declared[i].symbol.column
            })
            longname = ent_longname
```

Two rules are used in the listener of this file. One is “CompilationUnit” which checks if there no name of a package in this file, it will add just one reference with a line and a column of 0 and 1 (here where scope is a file, “None” is defined but in the main “None” is replaced by the file’s entity. Also “ent” which is the entity’s reference is completed in the “main”).

If there is a package declaration, The “Identifier” in the “qualifiedName” has each of the packages. Therefore, to make a loop we need to look at two things: 1) if it is the first package we need to make a reference between it and the file, 2) if it is not the first package we need to make a reference between it and the previous one.

Therefore, scope is the file or the previous package (if it is the file the reference in the dictionary would be None) and the entity of the reference is the package itself.
The line and column of the package is used to get the line and column of reference.

The use of “Declare Declarein Listener” in main.py

```
def addDeclareRefs(self, ref_dicts, file_ent):
```

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

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

        # Declare: kind id 192
        declare_ref = ReferenceModel.get_or_create(_kind=192, _file=file_ent,
_line=ref_dict["line"],
                                   _column=ref_dict["col"], _ent=ent, _scope=scope)

        # Declarein: kind id 193
        declarein_ref = ReferenceModel.get_or_create(_kind=193, _file=file_ent,
_line=ref_dict["line"],
                                   _column=ref_dict["col"], _scope=ent, _ent=scope)
```

For adding found references which are in the format of a dictionary to the database the “addDeclareRefs” method is used. This method for each “declare” is operated as follows:

If the “scope” type is “NoneType” it is treated as a file, otherwise, using the data in the dictionary the “entity” is created.

If the refrence’s entity is “None”, it will be considered as an unnamed package. Otherwise, it will be considered as a regular package and its “entity” is made.

For creating the references in the database, first we enter their “kind id” manually and then we add other information found to it.

Only the scope and entity of “declarein” are reversed and an integer is added to its “kind id”.

ImplementCouple and ImplementByCoupleBy references (implementCouple_implementbyCoupleby.py)

This reference happens when a class or enum implements another class or interface. After importing the JavaParserLabeledListener and JavaParserLabeled classes, We make a new listener class called ImplementCoupleAndImplementCoupleBy.
By searching the antlr4 parser rules, we found the token IMPLEMENTS to be included in two rules: ClassDeclaration and EnumDeclaration. So we consider these two rules in our listener to find these references.

```
class ImplementCoupleAndImplementByCoupleBy(JavaParserLabeledListener):
  """
  #Todo: Implementing the ANTLR listener pass for Java Call and Java Callby reference kind
  """
  implement = []
```

The “implement” python list will contain all the implement references we find on traversing the tree with this listener.

```
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, col] = str(ctx.start).split(",")[3].split(":")
    for myType in ctx.typeList().typeType():
```

```
        if myType.classOrInterfaceType():
            myType_longname = ".".join([x.getText() for x in
myType.classOrInterfaceType().IDENTIFIER()])
            self.implement.append({"scope_kind": "Class", "scope_name":
ctx.IDENTIFIER().__str__(),
                                   "scope_longname": scope_longname,
                                   "scope_parent": scope_parents[-2] if len(scope_parents) > 2
else None,
                                   "scope_contents": ctx.getText(),
                                   "scope_modifiers":

class_properties.ClassPropertiesListener.findClassOrInterfaceModifiers(ctx),
                                   "line": line,
                                   "col": col[:-1],
                                   "type_ent_longname": myType_longname})
```

In the method above which is written for classDeclaration rule, we first check whether or not the context has token IMPLEMENTS. If it does, it's a class that has an implement reference (the class is the scope entity of the implement reference).
Then we find this class's parents by calling a static method findParents implemented in ClassPropertiesListener which was described in previous pages and save it in the scope_parents variable. The longname of this class is obtained by joining the parents' names as well as this class's name in a single string separated by '.' character.
The line and column of the reference is found from ctx.start.
There may be multiple implement references in a single class declaration rule which are separated by a comma character. This is why we need to loop on the typelist().typeType() (that contains all the implemented entities) and get those which are of type classOrInterfaceType.
Each class or interface accessed in this loop can have multiple classOrInterfaceType rules, these are used to construct the longname of this entity.
In the end, we fill the fields found to an added element of "implements" list. The other fields added to this element are "scope" which is the name of this class found from the text in ctx.IDENTIFIER(), the "scope_kind" that is "Class" (used for finding the kind of this scope entity) and the modifiers.

The modifiers are returned by a static method findClassOrInterfaceModifiers which has already been discussed.
The method enterEnumDeclaration is the same as this with only two differences:

1. The context is of type JavaParserLabeled.EnumDeclarationContext.
2. The scope kind field is “Enum”.

```
def enterEnumDeclaration(self, ctx:JavaParserLabeled.EnumDeclarationContext):
  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, col] = str(ctx.start).split(",")[3].split(":")  # line, column
     for myType in ctx.typeList().typeType():
        if myType.classOrInterfaceType():
           myType_longname = ".".join([x.getText() for x in
myType.classOrInterfaceType().IDENTIFIER()])
           self.implement.append({"scope_kind": "Enum", "scope_name":
ctx.IDENTIFIER().__str__(),
                                  "scope_longname": scope_longname,
                                  "scope_parent": scope_parents[-2] if len(scope_parents) > 2
else None,
                                  "scope_contents": ctx.getText(),
                                  "scope_modifiers":

class_properties.ClassPropertiesListener.findClassOrInterfaceModifiers(
                                        ctx),
                                  "line": line,
                                  "col": col[:-1],
                                  "type_ent_longname": myType_longname})
```

The use of Implement and ImplementBy listener in main.py

```
# implement:
for file_address in files:
  try:
```

```
        file_ent = p.getFileEntity(file_address)
        listener = ImplementCoupleAndImplementByCoupleBy()
        listener.implement = []
        p.ParseAndWalk(listener, file_address)
        p.addImplementOrImplementByRefs(listener.implement, file_ent, file_address)
    except Exception as e:
        print("An Error occurred in file:" + file_address + "\n" + str(e) )
        continue
```

This part of code is the same as other references. The only difference is calling addImplementOrImplementByRefs at the end.

```
def addImplementOrImplementByRefs(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)

        implement_ref = ReferenceModel.get_or_create(_kind=188, _file=file_ent,
_line=ref_dict["line"],
                                                     _column=ref_dict["col"], _ent=ent, _scope=scope)
        implementBy_ref = ReferenceModel.get_or_create(_kind=189, _file=file_ent,
_line=ref_dict["line"],
                                                       _column=ref_dict["col"], _ent=scope, _scope=ent)
```

In this method, first we get or create the scope and entity's entity Id from the EntityModel class. Then we construct the implement and implementBy references.
To get (or create) the scope's id, we pass these arguments:

- _kind: kind of the scope is found by the findWithKeywords method described before.
- _name: is the value of this dictionary's scope_name.

- _parent: is the value of the dictionary's scope_parent. If it's none, the file is chosen as its parent.
- Longname and contents both from dictionaries scope_longname and scope_contents.

To get or create the entity's id, we used the function getImplementEntity depicted in the code below. As described before, the entity can be of type class or interface. This is why we first look for an interface with this longname, and if it's not found, we search for a class entity with the long name. The two methods getInterfaceEntity and getClassEntity are described in the previous sections.

```
def getImplementEntity(self, longname, file_address):
  ent = self.getInterfaceEntity(longname, file_address)
  if not ent:
     ent = self.getClassEntity(longname, file_address)
  return ent
```

After finding the entity Ids for the scope and entity, we can now create the two references:
Implement reference: kind id is 188 (we found this Id from the Kind table via an app named DB Browser.). The file is the file entity found previously and the line and column of this reference is in the dictionary. For scope and ent, we use the two scope and ent variables we found.
ImplementBy reference: Kind id is 189. The other fields are the same but the ent and scope are switched.

Create , Create by references (create_createby.py file)

Create , Createby will happen only when an object is created from a class, for example, a = new box().

In this file, we extend the Antlr listener so that we can use its methods. We make a "Create" array for saving all information.

We test the “tests.py” to find out when the “create” is happening, to see what the “understand” will show us. We get that “understand” gives us modifier (public , private) . Modifier (public , private) is returned from the “findmethodaccess” method. (To find modifier (public , private) it is necessary to go up in the rules in order to find the Class body Declaration).

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

    return parents,context
```

On the other hand, we need the types of the method (void , int , …) in which “new” has happend. To access the “method types” we implement “findmethodreturntypes” method. This method also returns the contents. (To obtain the method types , it is necessary to go up in the rules until we get to a rule whose “__name__” is “methoddeclarationcontext”). The reason “methoddeclarationcontext” is used is because of this: when we use “.__name__”, it adds the word “context” to the end of the name of the said rule. Then “.typeTypeorVoid” returns the method types.

```
def findmethodacess(self, c):
    parents = ""
    modifiers=[]
    current = c
    while current is not None:
        if "ClassBodyDeclaration" in type(current.parentCtx).__name__:
            parents=(current.parentCtx.modifier())
            break
```

```
        current = current.parentCtx
    for x in parents:
        if x.classOrInterfaceModifier():
            modifiers.append(x.classOrInterfaceModifier().getText())
    return modifiers
```

The condition “if ctx.creator().classCreatorRest” is because “understand” only returns create or createby when “new” is for a class not for an array.

We call “find parents” method from the class “classproperties” then we create “allrefs” variable for storing the array which we get back from “findparents” method. We set “refent” as the last element of “allrefs”. “Refent” is the “scope name” we are looking for.

To obtain “longname” we use “join” method for “allrefs” array and add a dot so that our format would be the same as “understand”.
In fact all “parents” are in“allrefs”. Finally, we fill the “create” array similar to “understand”.

```
create = []
def enterExpression4(self, ctx:JavaParserLabeled.Expression4Context):
    modifiers=self.findmethodacess(ctx)
    mothodedreturn,methodcontext=self.findmethodreturntype(ctx)

    if ctx.creator().classCreatorRest():
        allrefs= class_properties.ClassPropertiesListener.findParents(ctx)
#self.findParents(ctx)
        refent=allrefs[-1]
        entlongname=".".join(allrefs)
        print(refent , " refent")
        # khodi ke tush new shode
        [line, col] = str(ctx.start).split(",")[3].split(":")
        print(ctx.creator().createdName().getText()," In")
        print(".".join(allrefs[:-1]) + "." + ctx.creator().createdName().getText()," potential")

self.create.append({"scopename":refent,"scopelongname":entlongname,"scopemod
ifiers":modifiers,
```

```
                    "scopereturntype":mothodedreturn,"scopecontent":methodcontext,
                    "line":line,"col":col[:-
1],"refent":ctx.creator().createdName().getText(),
                    "scope_parent": allrefs[-2] if len(allrefs) > 2 else None,
                    "potential_refent":".".join(allrefs[:-1]) + "." +
ctx.creator().createdName().getText()})
```

The use of “Create Createby Listener” in main.py

First, we act on the basis of the explanations for the general codes in main.py.

```
# create
for file_address in files:
  file_ent = p.getFileEntity(file_address)
  listener = CreateAndCreateBy()
  listener.create = []
  p.ParseAndWalk(listener, file_address)
  p.addCreateRefs(listener.create, file_ent, file_address)
```

Then, for adding the found references, which are in the form of a dictionary, to the database, we use “addCreateRefs” method. This method operates as follows:

First for “ent” and “scope” we create a model in the database and fill them with the values found. For creating ent model we use “getCreatedClassEntity “ method. This method operates as follows:

First, using “class_potential_longname “ we check if the properties are returned from the “getclassproperties “ or not. If it is true we give the “getclassentity” method the “class_potential_longname” and get the entity obtained from potential long name (in case, the class which an object is

made from is local). Otherwise, we give “class_longname” to the said method (in case, the class which an object is made from is not local).

```
def getCreatedClassEntity(self, class_longname, class_potential_longname,
file_address):
  props = p.getClassProperties(class_potential_longname, file_address)
  if not props:
    return self.getClassEntity(class_longname, file_address)
  else:
    return self.getClassEntity(class_potential_longname, file_address)
```

Then, for adding the references to the database, we first enter the “kind id” manually, which we achieved with “db browser” program, then we add the other found information to it. “Createby” reference scope and entity are reversed and its “kind id” goes up by 1.

```
def addCreateRefs(self, ref_dicts, file_ent, file_address):
  for ref_dict in ref_dicts:
    scope = EntityModel.get_or_create(_kind=self.findKindWithKeywords("Method",
ref_dict["scopemodifiers"]),
                      _name=ref_dict["scopename"],
                      _type=ref_dict["scopereturntype"]
                      ,_parent=ref_dict["scope_parent"] if
ref_dict["scope_parent"]is not None else file_ent
                      , _longname=ref_dict["scopelongname"]
                      ,_contents=["scopecontent"])[0]
    ent = self.getCreatedClassEntity(ref_dict["refent"], ref_dict["potential_refent"],
file_address)
    Create = ReferenceModel.get_or_create(_kind=190, _file=file_ent,
_line=ref_dict["line"],
                        _column=ref_dict["col"], _scope=scope, _ent=ent)
    Createby = ReferenceModel.get_or_create(_kind=191, _file=file_ent,
_line=ref_dict["line"],
                        _column=ref_dict["col"], _scope=ent, _ent=scope)
```

3) Assessment

Because of the slow run speed, we couldn’t process projects having more Java files than the calculator app project (for example jfree project which has over 1000 Java files). The calculator project didn’t have two of the references we implemented, so we included the declare and declareIn references found and compared them to the understand database via a SQL select query:

```
Select referencemodel._line as line, referencemodel._column as col, file_ent._name as
filename,
scope._name as scope_name, scope._longname as scope_longname, scope._kind_id as
scope_kind,
ent._name as ent_name, ent._longname as ent_longname, ent._kind_id as ent_kind
from referencemodel inner join entitymodel scope on referencemodel._scope_id == scope._id
inner join entitymodel ent on referencemodel._ent_id == ent._id
inner join entitymodel file_ent on referencemodel._file_id = file_ent._id
where referencemodel._kind_id = 192 (یا 193) order by referencemodel._line,
referencemodel._column, file_ent._name, scope._name, ent._name;
```

The resulted tables are as follows:

line	col	filename	scope_name	scope_longname	scope_kind	ent_name	ent_longname	ent_kind
1	8	Main.java	Main.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\init\Main.java	1	com	com	72
1	8	basic_operation.java	basic_operation.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\basic_operation.java	1	com	com	72
1	8	fibonacci.java	fibonacci.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\fibonacci.java	1	com	com	72
1	8	integral.java	integral.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\integral.java	1	com	com	72
1	8	printLog.java	printLog.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\printLog.java	1	com	com	72
1	8	print_fail.java	print_fail.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_fail.java	1	com	com	72
1	8	print_success.java	print_success.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_success.java	1	com	com	72
1	8	println.java	println.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\println.java	1	com	com	72
1	12	Main.java	com	com	72	calculator	com.calculator	72
1	12	basic_operation.java	com	com	72	calculator	com.calculator	72
1	12	fibonacci.java	com	com	72	calculator	com.calculator	72
1	12	integral.java	com	com	72	calculator	com.calculator	72
1	12	printLog.java	com	com	72	calculator	com.calculator	72
1	12	print_fail.java	com	com	72	calculator	com.calculator	72

1	12	print_success.java	com	com	72	calculator	com.calculator	72
1	12	println.java	com	com	72	calculator	com.calculator	72
1	23	Main.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	basic_operation.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	fibonacci.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	integral.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	printLog.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	print_fail.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	print_success.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	println.java	calculator	com.calculator	72	app	com.calculator.app	72
1	27	Main.java	app	com.calculator.app	72	init	com.calculator.app.init	72
1	27	basic_operation.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	fibonacci.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	integral.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	printLog.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	print_fail.java	app	com.calculator.app	72	display	com.calculator.app.display	72
1	27	print_success.java	app	com.calculator.app	72	display	com.calculator.app.display	72
1	27	println.java	app	com.calculator.app	72	display	com.calculator.app.display	72

Declare references in our database

line	col	filename	scope_name	scope_longname	scope_kind	ent_name	ent_longname	ent_kind
1	8	Main.java	Main.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\init\Main.java	1	com	com	72
1	8	basic_operation.java	basic_operation.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\basic_operation.java	1	com	com	72
1	8	fibonacci.java	fibonacci.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\fibonacci.java	1	com	com	72
1	8	integral.java	integral.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\integral.java	1	com	com	72
1	8	printLog.java	printLog.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\printLog.java	1	com	com	72
1	8	print_fail.java	print_fail.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_fail.java	1	com	com	72
1	8	print_success.java	print_success.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_success.java	1	com	com	72
1	8	println.java	println.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\println.java	1	com	com	72
1	12	Main.java	com	com	72	calculator	com.calculator	72
1	12	basic_operation.java	com	com	72	calculator	com.calculator	72
1	12	fibonacci.java	com	com	72	calculator	com.calculator	72
1	12	integral.java	com	com	72	calculator	com.calculator	72
1	12	printLog.java	com	com	72	calculator	com.calculator	72
1	12	print_fail.java	com	com	72	calculator	com.calculator	72

1	12	print_success.java	com	com	72	calculator	com.calculator	72
1	12	println.java	com	com	72	calculator	com.calculator	72
1	23	Main.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	basic_operation.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	fibonacci.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	integral.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	printLog.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	print_fail.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	print_success.java	calculator	com.calculator	72	app	com.calculator.app	72
1	23	println.java	calculator	com.calculator	72	app	com.calculator.app	72
1	27	Main.java	app	com.calculator.app	72	init	com.calculator.app.init	72
1	27	basic_operation.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	fibonacci.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	integral.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	printLog.java	app	com.calculator.app	72	method	com.calculator.app.method	72
1	27	print_fail.java	app	com.calculator.app	72	display	com.calculator.app.display	72
1	27	print_success.java	app	com.calculator.app	72	display	com.calculator.app.display	72
1	27	println.java	app	com.calculator.app	72	display	com.calculator.app.display	72

Declare references in the Understand database

line	col	filename	scope_name	scope_longname	scope_kind	ent_name	ent_longname	ent_kind
1	8	Main.java	com	com	72	Main.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\init\Main.java	1
1	8	basic_operation.java	com	com	72	basic_operation.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\basic_operation.java	1
1	8	fibonacci.java	com	com	72	fibonacci.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\fibonacci.java	1
1	8	integral.java	com	com	72	integral.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\integral.java	1
1	8	printLog.java	com	com	72	printLog.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\printLog.java	1
1	8	print_fail.java	com	com	72	print_fail.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_fail.java	1
1	8	print_success.java	com	com	72	print_success.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_success.java	1
1	8	println.java	com	com	72	println.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\println.java	1
1	12	Main.java	calculator	com.calculator	72	com	com	72
1	12	basic_operation.java	calculator	com.calculator	72	com	com	72
1	12	fibonacci.java	calculator	com.calculator	72	com	com	72
1	12	integral.java	calculator	com.calculator	72	com	com	72
1	12	printLog.java	calculator	com.calculator	72	com	com	72
1	12	print_fail.java	calculator	com.calculator	72	com	com	72
1	12	print_success.java	calculator	com.calculator	72	com	com	72
1	12	println.java	calculator	com.calculator	72	com	com	72
1	23	Main.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	basic_operation.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	fibonacci.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	integral.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	printLog.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	print_fail.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	print_success.java	app	com.calculator.app	72	calculator	com.calculator	72

1	23	println.java	app	com.calculator.app	72	calculator	com.calculator	72
1	27	Main.java	init	com.calculator.app.init	72	app	com.calculator.app	72
1	27	basic_operation.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	fibonacci.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	integral.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	printLog.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	print_fail.java	display	com.calculator.app.display	72	app	com.calculator.app	72
1	27	print_success.java	display	com.calculator.app.display	72	app	com.calculator.app	72
1	27	println.java	display	com.calculator.app.display	72	app	com.calculator.app	72

DeclareIn references in our database

line	col	filename	scope_name	scope_longname	scope_kind	ent_name	ent_longname	ent_kind
1	8	Main.java	Main.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\init\Main.java	1	com	com	72
1	8	basic_operation.java	basic_operation.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\basic_operation.java	1	com	com	72
1	8	fibonacci.java	fibonacci.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\fibonacci.java	1	com	com	72
1	8	integral.java	integral.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\integral.java	1	com	com	72
1	8	printLog.java	printLog.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\printLog.java	1	com	com	72
1	8	print_fail.java	print_fail.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_fail.java	1	com	com	72
1	8	print_success.java	print_success.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_success.java	1	com	com	72
1	8	println.java	println.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\println.java	1	com	com	72
1	12	Main.java	com	com	72	calculator	com.calculator	72
1	12	basic_operation.java	com	com	72	calculator	com.calculator	72
1	12	fibonacci.java	com	com	72	calculator	com.calculator	72
1	12	integral.java	com	com	72	calculator	com.calculator	72
1	12	printLog.java	com	com	72	calculator	com.calculator	72
1	12	print_fail.java	com	com	72	calculator	com.calculator	72

line	col	filename	scope_name	scope_longname	scope_kind	ent_name	ent_longname	ent_kind
1	8	Main.java	com	com	72	Main.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\init\Main.java	1
1	8	basic_operation.java	com	com	72	basic_operation.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\basic_operation.java	1
1	8	fibonacci.java	com	com	72	fibonacci.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\fibonacci.java	1
1	8	integral.java	com	com	72	integral.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\integral.java	1
1	8	printLog.java	com	com	72	printLog.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\method\printLog.java	1
1	8	print_fail.java	com	com	72	print_fail.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_fail.java	1
1	8	print_success.java	com	com	72	print_success.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\print_success.java	1
1	8	println.java	com	com	72	println.java	D:\Term 7\Compiler\Final proj\github\OpenUnderstand\benchmark\calculator_app\src\com\calculator\app\display\println.java	1
1	12	Main.java	calculator	com.calculator	72	com	com	72
1	12	basic_operation.java	calculator	com.calculator	72	com	com	72
1	12	fibonacci.java	calculator	com.calculator	72	com	com	72
1	12	integral.java	calculator	com.calculator	72	com	com	72
1	12	printLog.java	calculator	com.calculator	72	com	com	72
1	12	print_fail.java	calculator	com.calculator	72	com	com	72
1	12	print_success.java	calculator	com.calculator	72	com	com	72
1	12	println.java	calculator	com.calculator	72	com	com	72
1	23	Main.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	basic_operation.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	fibonacci.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	integral.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	printLog.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	print_fail.java	app	com.calculator.app	72	calculator	com.calculator	72
1	23	print_success.java	app	com.calculator.app	72	calculator	com.calculator	72

1	23	println.java	app	com.calculator.app	72	calculator	com.calculator	72
1	27	Main.java	init	com.calculator.app.init	72	app	com.calculator.app	72
1	27	basic_operation.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	fibonacci.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	integral.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	printLog.java	method	com.calculator.app.method	72	app	com.calculator.app	72
1	27	print_fail.java	display	com.calculator.app.display	72	app	com.calculator.app	72
1	27	print_success.java	display	com.calculator.app.display	72	app	com.calculator.app	72
1	27	println.java	display	com.calculator.app.display	72	app	com.calculator.app	72

DeclareIn references in the Understand database

In the tables above all the fields considered in the query are the same for ours and the Understand's database.

4) Problems and Challenges

The problems and challenges in this project are as follows:

- **The association between the entities of two different files in a project are not considered**

Since each file is investigated separately to find the references, not considering defined packages in other files, the fields of the entity with a relationship to a separate package in the file will be incorrect/incomplete. For example, the entity's "longname" is not complete and doesn't have the name of the packages. Some of the entities which are completely defined in other files, will be "Unknown" in the created database.

- **Low Execution Speed**

Because of using a large number of "listener" for each file, the execution speed decreases greatly, such that after a few hours of file processing the benchmark execution will not be finished.

- **Omission of Whitespace in some Content's Field**

The reason that whitespaces are omitted is because some of the content fields of the entities such as class or interface entities are based on their context rules. But the file entity with contents does not have a problem.

5) Conclusions and Recommendations for Future Work

Using the presented codes we can find "implement, declare and create" from Java codes and investigate their entities characteristics.

The End