داکیومنت پروژه ی کامپایلر

گروه ۴

اعضای گروه:

شقایق مبشر ۹۷۵۲۲۰۰۴، پریسا علائی ۹۷۵۲۲۱۷۵، ستایش کولوبندی ۹۷۵۲۲۲۰۲

Contents

1. مقدمه

در این پروژه، افراد این گروه سعی در پیاده سازی متن باز از Understand Python API برای تجزیه و تحلیل کدهای منبع بوده اند. در واقع این کار توسعه یک پیاده‌سازی منبع باز Understand Python API، بوده است.
ابتدا بر پیاده سازی API برای برنامه های جاوا با استفاده از زبان های برنامه نویسی پایتون و ابزارهای کامپایلر مانند Antlr۴ کار شده است. و ساختارهایی را که توسط Understand برای تجزیه و تحلیل کدهای منبع استفاده می‌شود، بررسی شده است.
برای گروه ۴G باید تجزیه و تحلیلی را برای یافتن زیر مجموعه ای از انواع مراجع فهرست شده برای implement Couple_implementbyCouple by،Create Create by و declare declare in به همراه موجودیت های مربوطه خود ایجاد می شد.
مواردی که در پروژه استفاده میشود و بهتر است این جا معرفی شوند:

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

برای بخش پایگاه داده این پروژه از کتابخانه peewee و SQLite۳ استفاده شده است و بیشتر داده‌های جمع‌آوری‌شده شامل Entity, Reference است.
تقسیم کار برای انجام این پروژه به صورت این بوده است که هر نفر یک بخش لازم از سه بخش را انجام دهد:

۱. پریسا علایی بخش implement Couple _ implement by Couple by
۲. ستایش کلوبندی بخش Create Create by
۳. شقایق مبشر بخش declare declare in

در ادامه به ترتیب

- به روش پیشنهادی که شامل
  - کد اصلی در main، نحوه ی پیدا کرد
  - ست کردن های entity , parent
  - پیاده سازی declare declare
  - نحوه ی استفاده از declare declare in Listener
  - پیاده سازی implement Couple _ implement by Couple by
  - نحوه ی استفاده از implement Couple _ implement by Couple by in Listener
  - پیاده سازی Create Create by
  - نحوه ی استفاده از Create Create by in Listener
- ارزیابی

- که جواب های بدست آمده را با جواب هایی که understand به ما میدهد، مقایسه می شود.
- نتیجه گیری و کارهای آتی
- نشان دادن نتیجه کلی برنامه ی زده شده نوشته می شود.

می پردازیم

۲. روش پیشنهادی

کد کلی برای تمام قسمتها (**main.py**)

در این قسمت ابتدا دیتابیس پایه ساخته می‌شود و سپس روی تمام فایل‌های جاوای درون آدرس فولدر داده شده (path) برای هر کدام از رفرنس‌های پروژه یک حلقه می‌زند.

به ازای هر کدام از فایل‌ها در این حلقه ابتدا entity فایل برگردانده می‌شود (getFileEntity in Project class) و سپس listener مربوط به رفرنس ساخته می‌شود و در متد ParseAndWalk آن listener و آدرس فایل برای ساخت FileStream داده می‌شود. این متد درخت را walk کرده و listener با لیستی از خصوصیات رفرنس‌های پیدا شده در فایل پر می‌شود. سپس متد مخصوص به آن رفرنس برای پر کردن دیتابیس صدا زده می‌شود. همچنین در اینجا error handling صورت گرفته تا اگر یک فایل به مشکل خورد بتواند به اجرا ادامه دهد و دیگر فایلها را پردازش کند.

یافتن فیلدهای مربوط به یک کلاس یا اینترفیس (**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()
```

برای یافتن فیلدهای یک کلاس یا اینترفیس که در listener مربوط به آن تنها long name آن در دسترس بود از این دو listener استفاده می‌شود. (برای رفرنس‌های Implement/ImplementBy و (Create/CreateBy این دو listener تفاوت زیادی باهم ندارند، تنها به دلیل متفاوت بودن ruleهای مورد نیازشان (که ClassDeclaration باشد یا InterfaceDeclaration) این دو کلاس ClassPropertiesListener و InterfacePropertiesListener جدا ساخته شده‌اند.به همین دلیل تنها به شرح ClassPropertiesListener پرداخته شده است.

برای استفاده از listener ابتدا longname کلاس به صورت یک لیست که از استرینگ نام بالاترین parent تا نام خود کلاس را دارد داده می‌شود. همچنین یک class_properties نیز تعریف شده است که در صورتی که کلاس مورد نظر پیدا شود آن متغیر با یک دیکشنری از فیلدهای موردنیاز انتتی کلاس پر خواهد شد.

در این کلاس تنها یک rule بررسی شده است: classDeclaration.

توضیح enterClassDeclaration: با رسیدن به هر class declaration در درخت پارسر ابتدا چک می‌شود که در صورت پیدا شدن کلاس درست تا آن زمان ادامه متد اجرا نشود. اما اگر class_properties هنوز None باشد ابتدا نام کلاسی که به آن رسیده است (IDENTIFIER) با نام کلاس موردنظر که آخرین المان class_longname است مقایسه می‌شود. اگر نامشان برابر باشد باید چک شود که آیا parentهای این کلاس با کلاس موردنظر یکی است یا نه، به این دلیل که چند کلاس ممکن است با نام‌های یکسان در فایل موجود باشند، پس لازم است کلاسی که پرنت‌هایش با پرنت‌های کلاس موردنظر یکسان باشد را بیابیم. پرنت‌های کلاس را به کمک متد استاتیک findParents پیدا می‌کنیم که در قسمتی جدا توضیح داده شده است.

در صورتی که هم نام هم parentهای کلاس یکی باشد باید فیلدهای مورد نیاز برای ساخت انتیتی را پیدا کند. name, longname و parent از longname داده شده به دست می‌آیند. برای گرفتن modifierهای کلاس (مثلا public یا static بودن) از متد استاتیک findClassOrInterfaceModifiers استفاده کردم. این modifier برای پیدا کردن kind مرتبط با کلاس استفاده می‌شود. همچنین با گرفتن کل متن class declaration قسمت contents انتتی را به دست آوردیم.

یافتن **parent**های یک **entity** به کمک متد **findParents**

```
@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))
```

این متد context رول مربوط به آن انتتی را می‌گیرد و روی contextهای parent آن حلقه می‌زند. در این حلقه اگر contextهای پیدا شده مربوط به رول‌های ClassDeclaration،EnumDeclaration ، InterfaceDeclaration و یا AnnotationTypeDeclaration باشد که همه درون خود یک IDENTIFIER (نام آن کلاس / enum / interface / annotation type است) را دارند، متن IDENTIFIER به عنوان parent به لیست اضافه می‌شود. در پایان چون parentها از خود کلاس شروع شده و از پایین به بالاست، این لیست reverse شده است.

یافتن **modifier**های یک **entity** کلاس یا **interface** یا **enum** به کمک متد **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
```

با دقت در رول type declaration می‌یابیم که تمام modifierهای انواع انتتی‌های مورد نیاز ما برای پروژه می‌تواند با رسیدن به این رول و گرفتن classOrInterfaceModifier به دست آید. به همین دلیل مانند قبل parentهای این context را بالا رفته تا به یک type declaration برسد. سپس modifierهای درون آن به لیست modifiers اضافه شده است.

یافتن یا ایجاد مدل انتتی‌ها در کلاس **Project** (فایل **main.py**)

انتتی فایل (**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("processing file:",file_ent)
    return file_ent
```

این متد فیلدهای نام و longname انتتی فایل را با استفاده از آدرس آن به دست می‌آورد و همچنین با open کردن آن contents درونش را خوانده و تمام این فیلدها را بعلاوه kind id آن (که در دیتابیس ۱ است) برای گرفتن یا ساخت انتتی فایل به کار می‌برد.

انتتی پکیج (**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]
```

فیلدهای پکیج عبارت است از kind آن، نام و parent و longname. همچنین contents آن استرینگ خالی در نظر گرفته می‌شود.

انتتی پکیج بدون نام (getUnnamedPackageEntity)

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

فرق این متد با متد بالا در kind id و نام و longname است که دو مورد آخر (Unnamed Package) در نظر گرفته می‌شوند.

انتتی کلاس (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]
```

برای گرفتن انتتی یک کلاس به کمک آدرس فایل و نام کامل (longname) کلاس از این متد استفاده می‌شود. ابتدا به کمک این دو ClassPropertiesListener تلاش می‌کند این کلاس را در فایل پیدا کند. اگر نتواند این کار را بکند یعنی این کلاس در فایل تعریف نشده و لازم است انتتی از نوع unknown class باشد. در غیر این صورت باید نوع دقیق کلاس پیدا شود. برای این کار از متد findKindWithKeywords استفاده شده است که نوع انتتی ("Class") و modifierهای آن را میگیرد و مناسبترین kind را بازمیگرداند. سپس انتتی کلاس به کمک get_or_create به دست می‌آید. اگر در دیکشنری parent نداشته باشد یعنی parent آن خود فایل است.

انتتی اینترفیس (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:
    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]
```

برای گرفتن انتتی اینترفیس به کمک آدرس فایل و نام کامل (longname) اینترفیس از این متد استفاده می‌شود. ابتدا به کمک این دو InterfacePropertiesListener تلاش می‌کند آن را در فایل پیدا کند. اگر نتواند این کار را بکند از آنجا که اینترفیس از نوع unknown نداریم None برمیگرداند تا متدی که این متد را صدا زده به جای اینترفیس آن را کلاس در نظر بگیرد و اگر جایی تعریف نشده بود unknown class برگرداند. در غیر این صورت باید نوع دقیق اینترفیس پیدا شود. برای این کار از متد findKindWithKeywords استفاده شده است که نوع انتتی ("Interface") و modifierهای آن را میگیرد و مناسبترین kind را بازمیگرداند. سپس انتتی اینترفیس به کمک get_or_create به دست می‌آید. اگر در دیکشنری parent نداشته باشد یعنی parent آن خود فایل است.

متد کمکی findKindWithKeywords (فایل main.py)

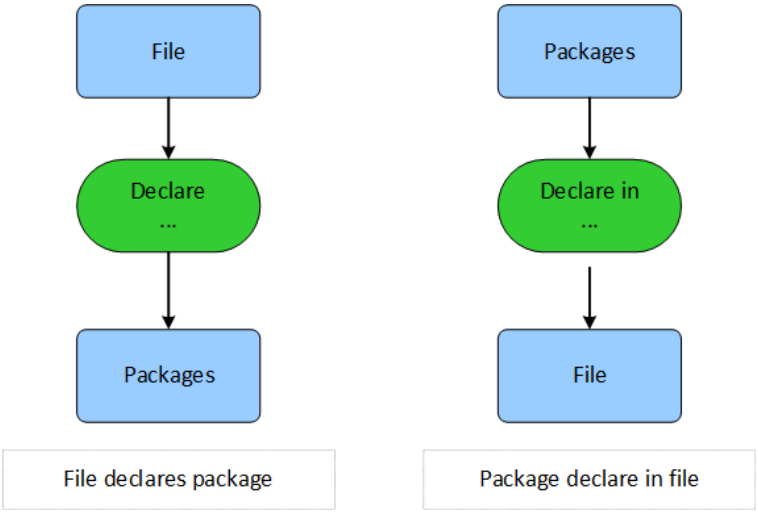
```
def findKindWithKeywords(self, type, modifiers):
  if len(modifiers) == 0:
    modifiers.append("default")
  leastspecific_kind_selected = None
  for kind in KindModel.select().where(KindModel._name.contains(type)):
    if self.checkModifiersInKind(modifiers, kind):
      if not leastspecific_kind_selected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

در این متد type منظور نوع اصلی (مثلا "Class" یا "Interface" یا "Enum" بودن نوع) و modifiers خصوصیات فرعی نوع (مثلا پابلیک بودن یا استاتیک بودن) را دارند. اگر modifiers خالی باشد یک default به آن اضافه میکنیم که نوع دیفالت آن پیدا شود.
سپس بین تمام kindهایی که درونشان type را دارند چک میکنیم و بین آن‌هایی که modifierها را هم در نامشان دارند kind با کوتاهترین نام را برمی‌گردانیم که احتمالا خصوصیات اضافی که در modifiers نیامده را ندارد. به طور مثال اگر modifiers تنها public داشته باشد و به دنبال یک کلاس باشیم، بین دو نوع Java Abstract Class Type و Public Member Java Class Type Public Member آن که کوتاه تر است را انتخاب می‌کند که نوع مورد نظر ماست.

رفرنس‌های declare و declarein (فایل declare_declarein.py)

رفرنس‌های declare با تعریف پکیج‌ها به وجود می‌آیند. تمام فایل‌ها یک پکیج را declare می‌کنند، و اگر نام پکیج در ابتدای فایل ذکر نشود این پکیج به صورت unnamed package به انتتی‌ها اضافه می‌شود. همچنین هر پکیج پکیجی که در ادامه نام آن آمده و با نقطه جدا شده است را declare می‌کند.




```
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
```

به این ترتیب دو rule در listener این فایل استفاده شده است. یکی CompilationUnit که چک شود اگر در این فایل نام پکیجی آورده نشده، تنها یک رفرنس با خط و ستون ۱ و ۰ اضافه کند. (در اینجا که اسکوپ فایل است None تعریف

شده اما در main خود انتتی فایل به جای آن قرار می‌گیرد. همچنین ent که منظور انتتی رفرنس است در main کامل می‌شود)
اگر package declaration وجود داشته باشد، IDENTIFIERهای درون qualifiedName آن تک تک پکیج‌ها را دارند. پس با حلقه زدن روی آن باید ۱. اگر اولین پکیج باشد یک رفرنس بین آن و فایل ایجاد کنیم و ۲. اگر پکیج‌های بعدی باشد بین آن و پکیج قبلی رفرنس را بسازیم.
پس scope فایل یا پکیج قبلی (که اگر فایل باشد در این دیکشنری رفرنس None گذاشته می‌شود) و entity رفرنس، آن پکیج است. برای گرفتن خط و ستون رفرنس از خط و ستون خود پکیج استفاده شده است. برای longname از استرینگ پکیج اول تا آن پکیج استفاده می‌شود.

استفاده از **Declare and Declarein Listener** در فایل **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)
```

برای اضافه کردن رفرنس‌های پیدا شده که به صورت دیکشنری هستند به دیتابیس از addDeclareRefs استفاده شده است. این متد برای هر دیکشنری از declareهای پیدا شده به صورت زیر عمل می‌کند:
اگر scope از نوع NoneType باشد آن را به عنوان فایل تلقی کرده و اگر نه به کمک اطلاعات داده شده در دیکشنری پکیج انتیتی آن را می‌سازد / می‌گیرد.
در صورتی که انتتی خود رفرنس None باشد آن را به عنوان پکیج unnamed تلقی می‌کند و در غیر این صورت آن را یک پکیج عادی در نظر می‌گیرد و انتتی آن را می‌سازد / می‌گیرد.

سپس برای ساخت رفرنس ها در دیتابیس، ابتدا آی دی kind آن را به صورت دستی (که با استفاده از برنامه db browser به آن دست پیدا کرده ایم) وارد می‌کنیم و سپس دیگر اطلاعاتی که تا اینجا جمع آوری کرده ایم را به آن اضافه می‌کنیم.
رفرنس declareIn تنها scope و entityاش برعکس است و آی دی kind آن یکی بعد از declare است.

رفرنس های implementCouple و implementbyCoupleby در فایل implementCouple_implementbyCoupleby.py

این رفرنس زمانی اتفاق می افتد که یک class یا enum می آید از implement استفاده می کند و یک اینترفیس یا کلاس را پیاده سازی می کند.
در این قسمت نیاز است که در ابتدا JavaParserLabeledListener وJavaParserLabeled را از فایل‌های گرامر جاوا در این متد ایمپورت کنیم.

در این فایل ما نیاز داریم که IMPLEMENTS ها را بدست بیاوریم. وقتی این استرینگ را در فایل های متعلق به جاوا گشتیم و در ClassDeclaration و EnumDeclaration آن را یافتیم . پس نیاز است که برای هردو آن ها IMPLEMENTS را چک کنیم و موارد خواسته شده را بیابیم.

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

در این قسمت یک لیست تعریف کردیم که همه ی موارد خواسته شده را در implement میریزیم .

```
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})
```

در این قسمت یک تابع صدا زدیم که وارد classDeclaration می شود و موارد مربوط به آن را بررسی می کند.
در ابتدا چک میکنیم که اگر در ctx کلمه ی implement باشد، کارهای لازم را انجام دهد. در خط بعد برای scope ها نیاز داریم که parents آن ها را بیابیم. با کمک تابعی که در فایل class_properties و در کلاس ClassPropertiesListener و در متد static ای به نام findParents، همه ی parent های مربوط به این scope در scope_parents ذخیره می شود.
در ادامه نیاز است که برای scope longname ها را نیز بدست آوریم که این کار در متد findParents، به ما برمی گردد. اگر تعداد خانه های scope_parents فقط یک باشد، Scope longname همان یک خانه میشود و در غیر این صورت همه ی خانه ها را یک '.' بهم وصل می کنیم و آن را به scope longname می دهیم.
برای گرفتن خط و ستون از ctx.start استفاده کردیم و جوابی را که می داد split و خانه هایی که نیاز داشتیم را برداشتیم.
چیزی که ClassDeclaration به می دهد، همه ی implement ها است که با کما از هم جدا شده اند پس نیاز داریم که یک loop بزنیم که در typelist هایی که دارد و typetype های آن ها بچرخد و type هایی که مربوط به classOrInterfaceType را بدست آوریم.Loop زده شده و شرط اولیه آن برای همین توضیحات داده شده است.
در ادامه برای entity هایی که داریم در این loop باید longname ها را بیابیم که میشود همه ی موجودیت هایی که در myType.classOrInterfaceType است که با join با "." به هم وصل شده اند.

در آخر همه ی مواردی را که پیدا کرده ایم را باید در لیستی که در بالا زده بودیم، اضافه کنیم.
در اینجا scope kind برابر class است چون ما در ClassDeclaration هستیم و اسم scope نیز میشود استرینگ ctx.IDENTIFIER() و scopelongname که در بالا بدست آوردیم و ذخیره کردیم را برای scope_longname می دهیم و برای parent این scope نیاز است که خانه ی ۲- لیست scope_parents را بدهیم چون خانه ی آخر خود scope است، این زمانی درست است که طول این لیست بیشتر از ۲ باشد در غیر این صورت برای این scope parent آن none است. برای پیدا کردن modifiers های این scope از فایل class_properties و از کلاس ClassPropertiesListener و از متد static ای به نام findClassOrInterfaceModifiers استفاده کرده ایم که در بالا توضیح آن داده شده است. در ادامه برای line مقدار بدست آورده شده در بالا را داده ایم و برای col تا خانه ی یکی مانده به آخر چیزی که بدست آوردیم زیرا یک "]" بیشتر داشت و ما به آن نیازی نداشتیم. و در آخر برای type_ent_longname، مقدار ذخیره شده ی myType_longname را داده ایم.
و به پایان این متد می رسیم.

برای متد enterEnumDeclaration درست مثل بالا عمل کرده ایم با دو تفاوت در کد :

۱. از ctx:JavaParserLabeled.EnumDeclarationContext که مربوط به enum است استفاده شده است.

۲. "scope_kind": "Enum"" است چون در قسمت enum هستیم.

```
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})
```

استفاده از Listener implement and implementBy در فایل 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
```

در این جا با توجه به توضیحات بالا و موارد توضیح داده شده است، عمل می کنیم و در آخر متد addImplementOrImplementByRefs را صدا می زنیم.

```
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)
```

در اینجا برای رفرنس هایی که داریم یک scope،entity، یک implement reference و implementBy reference می سازیم.
در ابتدا برای ساختن scope، از EntityModel با فیلدهایی که دارد یک مدل می سازیم. نحوه ی پرکردن فیلد :
برای kind از متد findKindWithKeywords استفاده کردیم که در بالا توضیح آن داده شده است.
برای name،longname،contents از موارد ذکر شده در بالا که در یک لیست ذخیره کرده بودیم استفاده کردیم.
برای قسمت parents نیز همینطور با این تفاوت که یک شرط برای آن قرار داده ایم که اگر این scope هیچ parent ای نداشت، parent آن اسم فایل اصلی جاوا می شود. در آخر متد get_or_create به ما یک آرایه برمی گرداند که ما خانه ی صفر آن را نیاز داریم.
برای ساختن entity متد getImplementEntity را صدا زده ایم که در آن طبق عکس کد پایین، دو حالت برای entity داریم ، یا class است یا interface است. در ابتدا ent را برابر خروجی متد getInterfaceEntity میگذاریم که در صورتی که entity آن interface باشد مقدار آن را به ent می دهیم ولی اگر از نوع interface نباشد مقدار ent برابر none است و وارد ایف میشود و با کمک متد getClassEntity مقدار آن داده می شود و در

آخر مقدار مقدار ent را برمی گردانیم. (دو متد getClassEntity و getInterfaceEntity در بالا توضیح داده شده اند.)

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

در ادامه باید refrence هایی که برای implement, implementBy هستند را بسازیم . نحوه ی مقدار دهی پارامترها برای implement reference :
برای مقدار kind از عدد ثابت ۱۸۸ استفاده کردیم که این عدد را بررسی دیتابیس با برنامه DB Browser پیدا کردیم و مقدار file را مقدار پاس داده شده و بدست آمده که در بالا توضیح داده ایم می گذاریم برای line و column از دو مقداری که در ابتدا توضیح داده ام استفاده میکنیم و در آخر برای مقدار ent, scope از دو مقدار بالایی که بدست آوردیم، استفاده می کنیم.
برای ساختن refrence برای implementBy مثل بالا عمل می کنیم با سه تفاوت :
برای عدد kind عدد ثابت ۱۸۹ را داده ایم و مقادیر scope و ent را باهم عوض می کنیم .

رفرنس های Create , Createby (فایل create_createby.py)

create , created by تنها داخل یک متد زمانی که یک object از یک کلاس ساخته می شود اتفاق می افتد.
برای مثال : a = new box ()
داخل این فایل listener خود انتلر را extend میکنیم که بتوانیم از متد هایش استفاده کنیم. آرایه create را برای ذخیره ی همه ی اطلاعات ساختیم .
زمانی که داخل فایل tests.py تست کردیم که ببینیم وقتی create اتفاق می افتد understand چه چیزهایی را به ما نشان می دهد دیدیم که داخل understand برای create به modifier (public private) نیاز داریم که این را با استفاده از متد find method access به دست آوردیم (برای پیدا کردن modifier باید به بالا برویم تا به rule ای برسیم که class body Declaration باشد و سپس از روی آن modifier ها را به دست بیاوریم)

```
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
```

از طرفی به تایپی که متدی که داخلش new اتفاق افتاده نیاز داریم یعنی void یا int یا .. برای به دست آوردن این موضوع متد find method return types را پیاده سازی کردیم که البته این متد contents را هم به عنوان خرجی برمیگرداند. (برای به دست آوردن تایپی که یک متد برمیگرداند باید بین rule ها بالا می رفتیم تا به rule ای برسیم که در rule method declaration context باشد چون داخل methoddeclarationcontext آن __name__. واقع داخل rule methoddeclaration (وقتی از __name__ استفاده میکنیم کلمه ی context به آخر نام rule اضافه می شود.) و سپس typeTypeorVoid. تایپی که آن متد ریترن میکند را برمی گرداند.

```
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
```

شرطی که بعد از آن گذاشتیم یعنی شرط ctx.creator().classCreatorRest برای این است که new فقط مربوط به کلاس باشد نه یک آرایه چون understand فقط زمانی create , createby را تشخیص میدهد که new فقط برای class اتفاق بیفتد.
سپس متد find parents را از داخل کلاس class properties صدا زدیم و refent را آخرین خانه ی آرایه ای که از متد find parents گرفتیم قرار دادیم (یعنی scope name) برای به دست آوردن longname روی آرایه ای که از متد گرفتیم که در واقع همه ی parent ها داخل آن قرار دارد join زدیم و دات گذاشتیم که فرمتمان شبیه فرمت understand شود و در آخر لیست create را مطابق با Understand پر کردیم.

```
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,"scopemodifiers":
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()})
```

استفاده از **Listener Create Createby** در فایل **main.py** :

ابتدا با توجه به توضیحات قسمت کد کلی برای تمام قسمتها (main.py) عمل می کنیم.

```
# 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)
```

سپس برای اضافه کردن رفرنس‌های پیدا شده که به صورت دیکشنری هستند به دیتابیس از addCreateRefs استفاده شده است. این متد برای هر دیکشنری ازcreate های پیدا شده به صورت زیر عمل می‌کند:
ابتدا برای ent , scope مدلی در دیتابیس می سازیم و آنها را با مقادیر به دست آمده پر میکنیم.
برای ساخت مدل ent از متد getCreatedClassEntity استفاده میکنیم که این متد به صورت زیر عمل میکند:
ابتدا با استفاده از class_potential_longname چک میکنیم که ببینیم خصوصیات آن از داخل متد getclassproperties به دست می آید یا نه اگر به دست آمد به متد getclassentity خود class_potential_longname را میدهیم وentity ای را می گیریم که با potential long name به دست آمده (درصورتی که class ای که از آن object ای ساخته شده است لوکال باشد) در غیر اینصورت class_longname را به متد ذکر شده میدهیم.(در صورتی که class ای که object ای از آن ساخته شده لوکال نباشد)

```
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)
```

سپس برای ساخت رفرنس ها در دیتابیس، ابتدا آی دی kind آن را به صورت دستی (که با استفاده از برنامه db browser به آن دست پیدا کرده ایم) وارد می‌کنیم و سپس دیگر اطلاعاتی که تا اینجا جمع آوری کرده ایم را به آن اضافه می‌کنیم.
رفرنس createby تنها scope و entity اش برعکس است و آی دی kind آن یکی بعد از create است.

```
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)
```

۳. ارزیابی

به دلیل سرعت بسیار کند کد ما در اجرا، نتوانستیم روی پروژه های با تعداد فایل بیشتر از پروژه calculator_app دیتابیس بسازیم. این پروژه هم به طور کلی دوتا از رفرنس های ما را نداشت اما declare و declareIn زیادی داشت.
برای این دو رفرنس یک کوئری SQL به صورت زیر نوشتیم:

```
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;
```

نتایج این کوئری را برای دو رفرنس میبینید:

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

نتایج declare دیتابیس ما

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

نتایج declare دیتابیس Understand

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

نتایج declareBy دیتابیس ما

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

نتایج declareBy دیتابیس understand

در جداول نشان داده شده تمام فیلدهایی که توسط query گرفته ایم با هم مطابقت دارند.

۴. مشکلات و چالش ها

- ### در نظر نگرفتن ارتباط موجودیت‌های دو فایل متفاوت در یک پروژه

به دلیل بررسی جداگانه هر فایل برای پیدا کردن رفرنس ها و در نظر نگرفتن پکیج های تعریف شده در فایل های دیگر، فیلد بعضی از entityها که ربطی به پکیجی جدا از این فایل داشته اند ناقص/ اشتباه است (به طور مثال longname آن entity کامل نیست و نام پکیج‌ها را ندارد.) و بعضی از موجودیت ها که به طور کامل در فایل دیگری تعریف شده اند در دیتابیس ساخته شده از نوع Unknown در آمده اند.

- ### سرعت اجرای پایین

به دلیل استفاده از تعداد زیادی listener برای هر فایل، سرعت پردازش فایلهای پروژه‌های benchmark به شدت پایین آمد، به طوری که بعد از چند ساعت پردازش فایل های benchmark به پایان نرسید.

- ### حذف شدن **whitespace** از بعضی فیلدهای **contents**

بعضی از موجودیت‌ها مانند موجودیت‌های کلاس یا interface، فیلد contentsشان از متن context رول مربوط به آنهاست و به همین دلیل whitespaceهایش حذف شده است. اما به طور مثال موجودیت فایل دارای contents درست است.

۵. نتیجه گیری و کارهای آتی

توسط کدهای ارائه شده میتوان سه رابطه declare،implement و create را از کدهای جاوا به خوبی تجزیه کرد و ارتباط و خصوصیات موجودیت‌های آن را بررسی کرد.

پایان