دانشکده مهندسی کامپیوتر

گزارش فاز دوم پروژه کامپایلر - گروه ۶

اعضای گروه

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

تیر ۱۴۰۱

فهرست مطالب:

تسک‌ها:

تسک‌های گروه ما به شرح زیر بود:

1 - CountDeclMethodAll
2 - CountDeclMethodDefault
3 - CountDeclMethodPrivate
4 - countDeclMethodProtected

تقسیم‌بندی:

نحوه انجام کار به این صورت بود که ۲ تسک اول را خانم باوان دیوانی آذر و بیان دیوانی اذر انجام دادند و ۲ تسک بعدی را خانم پارمیدا مجمع صنایع و زهرا مومنی نژاد انجام دادند. البته تمرکز بیشتر هر نفر به این صورت بوده و همگی در انجام همه‌ی موارد نقش داشتند.

روش انجام:

در فاز ۲ ما باید خروجی موارد بالا طبق دیتابیس understand و api های آن را با خروجی موارد بالا طبق دیتابیس خودمان و توابعی که برای کوئری زدن می‌زنیم مقایسه کنیم. همان طور که می‌دانیم دیتابیس ذکر شده در فایل main ایجاد شده و طبق فایل‌های پیاده‌سازی شده در فاز ۱ که در پوشه‌ی analysis passes هستند، پر می‌شود.

حال مفهوم تسک‌ها و نحوه پیاده‌سازی آن‌ها را بررسی می‌کنیم.

- **تسک اول – CountDeclMethodAll**

ابتدا مفهوم این بخش را شرح می‌دهیم.

منظور از این قسمت تعداد همه‌ی توابع یک فایل می‌باشد. که شامل تمام توابع خود فایل و تمام توابع فایل‌هایی که فایل ما از آن ارث‌بری کرده است می‌باشد. برای فهم بهتر مثال زیر را بررسی می‌کنیم:

یک کلاس employee داریم و یک کلاس manager که از آن ارث‌بری می‌کند. حال می‌خواهیم تعداد توابع کل manager را محاسبه کنیم. برای اینکار باید هم توابع خود فایل manager را درنظر بگیریم، و هم توابع پدر آن را. یعنی کلاس employee.

فایل employee که یک کلاس دارد به شرح زیر است:

```
public class employee {

   static long employeeId;
   static String employeeName;
   static String employeeAddress;
   static long employeePhone;
   static double basicSalary;
   static double specialAllowance = 250.80;
```

```
    static double hra = 1000.50;

    default static long getEmployeeId() {
        return employeeId;
    }
    protected static void setEmployeeId(long Id) {
        employee.employeeId = Id;
    }

    protected static String getEmployeeName() {
        return employeeName;
    }
    default static void setEmployeeName(String Name) {
        employee.employeeName = Name;
    }
    protected static String getEmployeeAddress() {
        return employeeAddress;
    }
    protected static void setEmployeeAddress(String address) {
        employee.employeeAddress = address;
    }
    default static long getEmployeePhone() {
        return employeePhone;
    }
    protected static void setEmployeePhone(long phone) {
        employee.employeePhone = phone;
    }

    private static void calculateSalary () {
        double salary = basicSalary + (basicSalary * specialAllowance/100) +
basicSalary * hra/100;
        System.out.println(salary);
    }

    private double calculateTransportAllowance () {
        double transportAllowance = 10/100*basicSalary;
        return transportAllowance;
    }

}
```

تعداد کل تابع‌های این فایل ۱۰ تاست. حال فایل **manager** را بررسی کنیم:

```
public class Manager extends employee {
    protected static double getBasicSalary() {
        return basicSalary;
    }
    protected static void setBasicSalary(double basicSalary) {
        Manager.basicSalary = basicSalary;
    }
```

```
@Override
private double calculateTransportAllowance () {
   double transportAllowance = 15 * basicSalary/100;
   return transportAllowance;
}
```

تعداد کل تابع‌های این فایل هم ۳ تاست. اما چون این کلاس از کلاس **employee** ارث‌بری کرده است، باید تعداد توابع کلاس پدر را هم در تعداد توابع کل **manager** حساب کنیم. پس **CountDeclMethodAll** برابر با **۱۳** تا خواهد بود.

حال به نحوه پیاده‌سازی این بخش می‌پردازیم:

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه **metrics** ایجاد کردیم.

در ابتدا فایل‌های مورد نیاز و توابع مورد نیاز را **import** کردیم و بعد هم تابع **walk** را برای پیمایش روی درخت تعریف کردیم.

```
from oudb.api import open
from oudb.models import EntityModel, KindModel
from antlr4 import *
from gen.javaLabeled.JavaParserLabeled import JavaParserLabeled
from gen.javaLabeled.JavaLexer import JavaLexer
from analysis_passes.extend_listener_g6 import ExtendListener


def Walk(reference_listener, parse_tree):
    walker = ParseTreeWalker()
    walker.walk(listener=reference_listener, t=parse_tree)
```

سپس یک تابع به نام **count_decl_method_all** ایجاد کردیم.

در آن دو دیکشنری به نام‌های **class_methods** و **extend_class_names** تعریف کردیم. که به ترتیب اولی نام فایل و تعداد توابع کل و دومی نام کلاس و نام کلاس‌هایی که کلاس ما از آن‌ها ارث‌بری می‌کند را نگه می‌دارد.

یک لیست به نام **files** هم داریم که فایل‌ها را نگه می‌دارد.

سپس یک حلقه **for** زدیم که چک کنیم در بین **entity model** ها هر کدام که فایل بود را به لیست خود اضافه می‌کنیم. از این طریق به فایل‌های پروژه دسترسی پیدا می‌کنیم. و اگر **class** بود دیکشنری **class_methods** را با **key** با ارزش اسم **entity** و **value** با ارزش ۰ اپدیت می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
def count_decl_method_all(dbname):
    open(dbname)
    class_methods = {}
    files = []
    extends_class_names = {}
    # get files names
    for ent_model in EntityModel.select():
        if ent_model._kind_id == 1:
            files.append(ent_model._longname)
        if "Class" in ent_model._kind._name:
            class_methods[ent_model._name]=0
```

در مرحله‌ی بعدی، یک حلقه‌ی **for** دیگر برای کال کردن **listener** زده شده با عنوان **extendListener** استفاده کردیم. که این **listener** را در فایل **extend_listener_g6** تعریف کردیم و اکنون باهم عملکرد آن را بررسی کرده و به ادامه این بخش بازمی‌گردیم.

در این فایل ابتدا دو فایل **parser** و **listener** مورد نیاز خود را **import** کردیم و یک کلاس جدید برای **listener** خود تعریف کردیم. که این کلاس ۲ تا **property** دارد. یک **class_name** که اسم کلاس است و یک دیکشنری به نام **refers** که روابط و **parent** ها و ارث‌بری‌ها را نگه می‌دارد. همچنین دو تابع **getter** برای دریافت مقادیر این دو متغیر تعریف کردیم.

سپس قسمت اصلی اینجاست که ما تابع **enterClassDeclaration** را **override** کردیم. برای چه؟ برای اینکه پیدا کنیم هر کلاس چه کلاس‌هایی را **extend** کرده است و آن‌ها را در دیکشنری خود ذخیره کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
from gen.javaLabeled.JavaParserLabeledListener import
JavaParserLabeledListener
from gen.javaLabeled.JavaParserLabeled import JavaParserLabeled

class ExtendListener(JavaParserLabeledListener):

    def __init__(self):
        self.class_name = None
        self.refers = {}


    def get_refers(self):
        return self.refers
```

```
def get_class_name(self):
        return self.class_name
def enterClassDeclaration(self,
ctx:JavaParserLabeled.ClassDeclarationContext):
        self.class_name = ctx.IDENTIFIER().getText()
        if ctx.getChild(2).getText() == "extends":
            childs=ctx.getChild(3).getChildren()
            for c in childs:
                if not self.refers.__contains__(self.class_name):
                    self.refers[self.class_name] = []
                self.refers[self.class_name].append(c.getText())
```

توضیح و عملکرد **extendListener** را با هم دیدیم. به ادامه‌ی توضیح کلاس **count_decl_method_all** برمی‌گردیم. همانطور که بیان شد، از یک حلقه‌ی **for** استفاده می‌کنیم تا روی تمامی فایل‌های پروژه این **extendListener** را صدا بزنیم و بعد روی درخت پیمایش می‌کنیم تا تابع **override** شده انجام شود و دیکشنری **refers** در **extendListener** پر شود. بعد آن را **get** می‌کنیم و در دیکشنری **extend_class_names** ذخیره می‌کنیم. تا اینجا لیست ارث‌بری‌ها را بدست آوردیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
# get parent names
for file_address in files:
    try:
        file_stream = FileStream(file_address, encoding='utf8')
        lexer = JavaLexer(file_stream)
        tokens = CommonTokenStream(lexer)
        parser = JavaParserLabeled(tokens)
        parse_tree = parser.compilationUnit();
    except Exception as e:
        print("An Error occurred in file:" + file_address + "\n" +
str(e))
        continue
    try:
        listener = ExtendListener()
        Walk(listener,parse_tree)
        extends_class_names.update(listener.get_refers())
    except Exception as e:
        print("An Error occurred for reference implement in file:" +
file_address + "\n" + str(e))
```

حال می‌خواهیم توابع را پیدا کنیم. برای همین از یک حلقه **for** استفاده می‌کنیم. و روی **entity model** ها پیمایش می‌کنیم و درصورتی که **kind name** آن برابر **Method** باشد، دیکشنری **class_method** خود را اپدیت می‌کنیم. به این صورت که **key** اسم **parent** باشد و **value** هم یک عدد است که تعداد توابع را نشان می‌دهد. و اگر **initial** شده باشد هر بار **count** آن درصورت یافتن **Method** جدید یکی زیاد می‌شود.

کد این قسمت توضیح داده شده به صورت زیر است:

```
# get class methods number
for ent_model in EntityModel.select():

    if "Method" in ent_model._kind._name:
        exists = class_methods.get(ent_model._parent._name, -1)
        if exists == -1:
            class_methods[ent_model._parent._name] = 1
        else:
            class_methods[ent_model._parent._name] += 1
```

در آخر هم از یک حلقه **for** استفاده می‌کنیم تا روی **extend_class_names** پیمایش کنیم و یک لیست **visited** و یک **stack** تعریف می‌کنیم. برای اینکه بیایم چک کنیم که کلاسی که هر کلاس از آن ارث بری کرده است آیا خودش هم از کلاسی ارث‌بری کرده است. و بعد آن را به **class_methods** اضافه می‌کنیم. و در آخر تابع آن را **return** می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
for cm in class_methods:
    stack=[]
    visited=[]
    temp = cm
    while extends_class_names.__contains__(temp):
        for t in extends_class_names[temp]:
            if not visited.__contains__(t):
                stack.append(t)
                visited.append(t)
        temp = stack.pop()
    for v in visited:
        class_methods[cm] += class_methods[v]


return class_methods
```

- تسک دوم – **CountDeclMethodDefault**

ابتدا مفهوم این بخش را شرح می‌دهیم.

منظور از این بخش تعداد تمام توابعی هستند که با کلیدواژه‌ی default تعریف شدند. مثالی از این نوع تابع بصورت زیر است:

```
default void show()
{
     System.out.println("Default Method Executed");
}
```

حال به نحوه پیاده‌سازی این بخش می‌پردازیم:

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه metrics ایجاد کردیم.

سپس از api های oudb مورد open و از model ها مورد EntityModel را import کردیم. و بعد یک تابع به اسم تسک ایجاد کردیم که به عنوان ورودی مسیر دیتابیس ما را می‌گیرد. و بعد با استفاده از open آن را می‌خواند. یک دیکشنری به اسم class_methods نیز تعریف کردیم که اسم هر کلاس را به عنوان کلید و تعداد توابع default آن را به عنوان value دربردارد.

چون می‌خواهیم توابع یک کلاس را بررسی کنیم، ابتدا روی entity model ها با یک حلقه for پیمایش می‌کنیم و اگر kind name برابر با class بود دیکشنری را با اسم entity model به عنوان key و 0 به عنوان value آن initial می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
from oudb.api import open
from oudb.models import EntityModel


def count_decl_method_default(db_path):
    open(db_path)
    class_methods = {}

    for ent_model in EntityModel.select():
        if "Class" in ent_model._kind._name:
            class_methods[ent_model._name]=0
```

سپس دوباره برای پیدا کردن توابع default از یک حلقه for و از یک شرط استفاده می‌کنیم. به این صورت که چک می‌کنیم درون kind name آن entity model هم واژه‌ی default باشد و هم واژه‌ی method. سپس اگر اسم parent آن entity model را درون دیکشنری خود داشتیم، تعداد توابع آن کلاس را اپدیت می‌کنیم و یکی زیاد می‌کنیم. و در آخر تابع هم این دیکشنری را return می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
for ent_model in EntityModel.select():
    if "Default" in ent_model._kind._name and "Method" in
ent_model._kind._name:
        exists = class_methods.get(ent_model._parent._name, -1)
        if exists == -1:
            class_methods[ent_model._parent._name] = 1
        else:
            class_methods[ent_model._parent._name] += 1
return class_methods
```

- تسک سوم – **CountDeclMethodPrivate**

ابتدا مفهوم این بخش را شرح می‌دهیم.

منظور از این بخش تعداد تمام توابعی هستند که با کلیدواژه‌ی private تعریف شدند. مثالی از این نوع تابع بصورت زیر است:

```
private double calculateTransportAllowance () {
   double transportAllowance = 15 * basicSalary/100;
   return transportAllowance;
}
```

حال به نحوه پیاده‌سازی این بخش می‌پردازیم:

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه metrics ایجاد کردیم.

سپس از api های oudb مورد open و از model ها مورد EntityModel را import کردیم. و بعد یک تابع به اسم تسک ایجاد کردیم که به عنوان ورودی مسیر دیتابیس ما را می‌گیرد. و بعد با استفاده از open آن را می‌خواند. یک دیکشنری به اسم class_methods نیز تعریف کردیم که اسم هر کلاس را به عنوان کلید و تعداد توابع private آن را به عنوان value دربردارد.

چون می‌خواهیم توابع یک کلاس را بررسی کنیم، ابتدا روی entity model ها با یک حلقه for پیمایش می‌کنیم و اگر kind name برابر با class بود دیکشنری را با اسم entity model به عنوان key و 0 به عنوان value آن initial می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
from oudb.api import open
from oudb.models import EntityModel

def count_decl_method_private(db_path):
    open(db_path)
    class_methods = {}

    for ent_model in EntityModel.select():
        if "Class" in ent_model._kind._name:
            class_methods[ent_model._name]=0
```

سپس دوباره برای پیدا کردن توابع **private** از یک حلقه **for** و از یک شرط استفاده می‌کنیم. به این صورت که چک می‌کنیم درون **kind name** آن **entity model** هم واژه‌ی **private** باشد و هم واژه‌ی **method**. سپس اگر اسم **parent** آن **entity model** را درون دیکشنری خود داشتیم، تعداد توابع آن کلاس را اپدیت می‌کنیم و یکی زیاد می‌کنیم. و در آخر تابع هم این دیکشنری را **return** می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
for ent_model in EntityModel.select():
    if "Private" in ent_model._kind._name and "Method" in
ent_model._kind._name:
        exists = class_methods.get(ent_model._parent._name, -1)
        if exists == -1:
            class_methods[ent_model._parent._name] = 1
        else:
            class_methods[ent_model._parent._name] += 1
return class_methods
```

- **تسک چهارم – CountDeclMethodProtected**

ابتدا مفهوم این بخش را شرح می‌دهیم.

منظور از این بخش تعداد تمام توابعی هستند که با کلیدواژه‌ی protected تعریف شدند. مثالی از این نوع تابع بصورت زیر است:

```
protected static void setBasicSalary(double basicSalary) {
   Manager.basicSalary = basicSalary;
}
```

حال به نحوه پیاده‌سازی این بخش می‌پردازیم:

ابتدا برای اینکار یک فایل پایتون متناسب با اسم تسک در پوشه metrics ایجاد کردیم.

سپس از api های oudb مورد open و از model ها مورد EntityModel را import کردیم. و بعد یک تابع به اسم تسک ایجاد کردیم که به عنوان ورودی مسیر دیتابیس ما را می‌گیرد. و بعد با استفاده از open آن را می‌خواند. یک دیکشنری به اسم class_methods نیز تعریف کردیم که اسم هر کلاس را به عنوان کلید و تعداد توابع protected آن را به عنوان value دربردارد.

چون می‌خواهیم توابع یک کلاس را بررسی کنیم، ابتدا روی entity model ها با یک حلقه for پیمایش می‌کنیم و اگر kind برابر با name بود دیکشنری را با اسم class به عنوان key و entity model به عنوان value آن initial می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
from oudb.api import open
from oudb.models import EntityModel


def count_decl_method_protected(db_path):
    open(db_path)
    class_methods = {}

    for ent_model in EntityModel.select():
        if "Class" in ent_model._kind._name:
            class_methods[ent_model._name]=0
```

سپس دوباره برای پیدا کردن توابع protected از یک حلقه for و از یک شرط استفاده می‌کنیم. به این صورت که چک می‌کنیم درون kind name آن entity model هم واژه‌ی protected باشد و هم واژه‌ی method. سپس اگر اسم parent آن entity model را درون دیکشنری خود داشتیم، تعداد توابع آن کلاس را اپدیت می‌کنیم و یکی زیاد می‌کنیم. و در آخر تابع هم این دیکشنری را return می‌کنیم.

کد این قسمت توضیح داده شده به صورت زیر است:

```
for ent_model in EntityModel.select():
    if "Protected" in ent_model._kind._name and "Method" in
ent_model._kind._name:
        exists = class_methods.get(ent_model._parent._name, -1)
        if exists == -1:
            class_methods[ent_model._parent._name] = 1
        else:
            class_methods[ent_model._parent._name] += 1
return class_methods
```

❋ خروجی گرفتن با استفاده از تابع‌های خودمان

برای خروجی گرفتن از ۴ تابعی که زدیم، یک فایل تست به نام **db_test_metrics_g6** ایجاد کردیم، که در آن ۴ تابع زده شده در ۴ فایل توضیح داده شده را **import** کردیم. یک تابع کلی به نام **test_all_metrics** زدیم که مسیر یک دیتابیس را به عنوان ورودی می گیرد و ۴ تابع برای پیدا کردن تعداد توابع **all** و **default** و **private** و **protected** را روی این ورودی صدا می‌زند.

کد این تابع بصورت زیر است:

```
from count_decl_method_all import count_decl_method_all
from count_decl_method_default import count_decl_method_default
from count_decl_method_private import count_decl_method_private
from count_decl_method_protected import count_decl_method_protected

def test_all_metrics(db_path):
    print("Our Results \n")
    print("All methods : ", count_decl_method_all(db_path))
    print("Default methods : ", count_decl_method_default(db_path))
    print("Private methods : ", count_decl_method_private(db_path))
    print("Protected methods : ",
count_decl_method_protected(db_path))

if __name__ == '__main__':
   test_all_metrics("..\\..\\benchmark2_database.oudb")
```

تابع را روی دیتابیس ساخته شده به نام **benchmark2_database.oudb** اجرا می‌کنیم و خروجی به صورت زیر است:

```
All methods :  {'employee': 10, 'inheritanceActivity': 3, 'Manager': 13, 'MiniTrainee': 16, 'Trainee': 13}
Default methods :  {'employee': 3, 'inheritanceActivity': 0, 'Manager': 0, 'MiniTrainee': 1, 'Trainee': 1}
Private methods :  {'employee': 2, 'inheritanceActivity': 0, 'Manager': 1, 'MiniTrainee': 1, 'Trainee': 1}
Protected methods :  {'employee': 5, 'inheritanceActivity': 2, 'Manager': 2, 'MiniTrainee': 1, 'Trainee': 1}

Process finished with exit code 0
```

❋ خروجی گرفتن از understand

برای خروجی گرفتن از ابزار understand یک فایل تست در پوشه‌ی oudb به نام tests.py ایجاد کردیم. ابتدا موارد لازم برای اجرای تست را import کردیم و صدا زدیم. یک تابع به نام test_understand_kinds تعریف کردیم و با استفاده از open توسط und.open دیتابیس موردنظر را باز کردیم. برای نگه‌داری خروجی ۴ نوع metric مان ۴ آرایه برای توابع all و default و private و protected ایجاد کردیم.

کد این قسمت بصورت زیر است:

```
import os
os.add_dll_directory(r"C:\Program Files\Scitools\bin\pc-win64")
from dotenv import load_dotenv
load_dotenv()
try:
    import understand as und
except ImportError:
    import understand as und

    print("Can not import understand")

def test_understand_kinds():
    db = und.open("..\\..\\src.und")
    und_all_results = {}
    und_default_results = {}
    und_private_results = {}
    und_protected_results = {}
```

سپس با استفاده از یک حلقه‌ی for روی entity هایی که class هستند پیمایش کردیم و از api آماده understand به نام metric استفاده کردیم و تعداد هر metric را بدست آوردیم. درنهایت هر ۴ مورد را print می‌کنیم.

کد این قسمت بصورت زیر است:

```
print("Understand Results \n")
for ent in db.ents('Java Class ~Unknown ~Unresolved'):
    ent_name = ent.name()
    all_methods =
ent.metric(['CountDeclMethodAll']).get('CountDeclMethodAll', 0)
und_all_results[ent_name] = all_methods

default_methods =
ent.metric(['CountDeclMethodDefault']).get('CountDeclMethodDefault', 0)
und_default_results[ent_name] = default_methods
private_methods =
ent.metric(['CountDeclMethodPrivate']).get('CountDeclMethodPrivate', 0)
```

```
und_private_results[ent_name] = private_methods

protected_methods =
ent.metric(['CountDeclMethodProtected']).get('CountDeclMethodProtected', 0)
und_protected_results[ent_name] = protected_methods

print("All methods : ", und_all_results)
print("Default methods : ", und_default_results)
print("Private methods : ", und_private_results)
print("Protected methods : ", und_protected_results)

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

تابع را روی دیتابیس ساخته شده توسط **understand** به نام **src.und** اجرا می‌کنیم و خروجی به صورت زیر است:

```
All methods :  {'Manager': 13, 'employee': 10, 'inheritanceActivity': 3, 'Trainee': 13, 'MiniTrainee': 16}
Default methods :  {'Manager': 0, 'employee': 3, 'inheritanceActivity': 0, 'Trainee': 1, 'MiniTrainee': 1}
Private methods :  {'Manager': 1, 'employee': 2, 'inheritanceActivity': 0, 'Trainee': 1, 'MiniTrainee': 1}
Protected methods :  {'Manager': 2, 'employee': 5, 'inheritanceActivity': 2, 'Trainee': 1, 'MiniTrainee': 1}

Process finished with exit code 0
```

مقایسه خروجی‌ها

❋ حال برای اثبات درستی توابعی که زدیم خروجی دو فایل تست را باهم مقایسه می‌کنیم:

خروجی ما :

```
All methods :  {'employee': 10, 'inheritanceActivity': 3, 'Manager': 13, 'MiniTrainee': 16, 'Trainee': 13}
Default methods :  {'employee': 3, 'inheritanceActivity': 0, 'Manager': 0, 'MiniTrainee': 1, 'Trainee': 1}
Private methods :  {'employee': 2, 'inheritanceActivity': 0, 'Manager': 1, 'MiniTrainee': 1, 'Trainee': 1}
Protected methods :  {'employee': 5, 'inheritanceActivity': 2, 'Manager': 2, 'MiniTrainee': 1, 'Trainee': 1}

Process finished with exit code 0
```

خروجی understand :

```
All methods :  {'Manager': 13, 'employee': 10, 'inheritanceActivity': 3, 'Trainee': 13, 'MiniTrainee': 16}
Default methods :  {'Manager': 0, 'employee': 3, 'inheritanceActivity': 0, 'Trainee': 1, 'MiniTrainee': 1}
Private methods :  {'Manager': 1, 'employee': 2, 'inheritanceActivity': 0, 'Trainee': 1, 'MiniTrainee': 1}
Protected methods :  {'Manager': 2, 'employee': 5, 'inheritanceActivity': 2, 'Trainee': 1, 'MiniTrainee': 1}

Process finished with exit code 0
```

✓ همانطور که می‌بینیم تعدادهای بدست آمده از توابعی که زدیم با خروجی understand **برابر** است.