# به نام خدا

تمرین سری چهارم
درس اصول طراحی کامپایلر
دکتر سعید پارسا

فرزان رحمانی
۹۹۵۲۱۲۷۱

## سوال اول

برای حل این سوال از گرامر زبان جاوا و antlr استفاده کردیم. همچنین دو ورودی تعریف کردیم که شامل سه کلاس دانشجو، دانشگاه و شهر هستند. در ورودی اول cycle وجود ندارد و کد بهینه هست. در ورودی دوم cycle موجود است. همچنین برای پیاده سازی از یک Listener شخصی سازی شده استفاده نمودیم. این کلاس از Listener تولید شده توسط جاوا ارث بری می‌کند. با مطالعه گرامر زبان جاوا پی بردیم که با استفاده از exitCompilationUnit، enterNormalClassDeclaration و enterNormalInterfaceDeclaration و استفاده از netwrokx می‌توانیم گراف کلاس ها را ساخته با کمک الگوریتم پیمایش درخت مانند dfs آن ها را تشخیص داده و نمایش دهیم. ما در اینجا از تابع nx.simple_cycles استفاده کردیم.
خروجی کد در دو حالت دارای cycle و بدون cycle در ادامه آمده است. همچنین ورودی ها و توابع اصلی را هم مشاده میکنید.

خروجی با cycle:

```
(base) E:\uni\7th term\fundamentals of compiler design\HWs\HW4\HW4_99521271>python main.py
The number of cycles detected in the Java code: 1
['University', 'City', 'Student']
```

خروجی بدون cycle:

```
(base) E:\uni\7th term\fundamentals of compiler design\HWs\HW4\HW4_99521271>python main.py
The number of cycles detected in the Java code: 0
```

ورودی با cycle:

```
// Class representing a University that is coupled with the City class
class University {
    private String ID;
    private String name;
    private City city;

    public University(String ID, String name, City city) {
        this.ID = ID;
        this.name = name;
        this.city = city;
    }
```

```java
}

// Class representing a Student that is coupled with the University class
class Student {
    private String FirstName;
    private String LastName;
    private University uni;

    public Student(String FirstName, String LastName, University uni) {
        this.FirstName = FirstName;
        this.LastName = LastName;
        this.uni = uni;
    }

    public Student(String FirstName, String LastName) {
        this.FirstName = FirstName;
        this.LastName = LastName;
    }

    public void AddUniversity(University uni) {
        this.uni = uni;
    }
}

// Class representing a City that is coupled with the Student class
class City {
    private String name;
    private Student student;

    public City(String name, Student student) {
        this.name = name;
        this.student = student;
    }
}

public class Main {
    public static void main(String[] args) {
        // Creating an instance of the Student class
        Student student = new Student("Farzan", "Rahmani");

        City city = new City("tehran", student);

        // Creating an instance of the University class
        University uni = new University("12345678", "IUST", city);
```

```
        // and associating Student with the University
        student.AddUniversity(uni);
    }
}
```

ورودی بدون cycle:

```
// Class representing a University that is coupled with the City class
class University {
    private String ID;
    private String name;
    private City city;

    public University(String ID, String name, City city) {
        this.ID = ID;
        this.name = name;
        this.city = city;
    }

}

// Class representing a Student that is coupled with the University class
class Student {
    private String FirstName;
    private String LastName;
    private University uni;

    public Student(String FirstName, String LastName, University uni) {
        this.FirstName = FirstName;
        this.LastName = LastName;
        this.uni = uni;
    }
}

// Class representing a City
class City {
    private String name;

    public City(String name) {
        this.name = name;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        City city = new City("tehran");

        // Creating an instance of the University class
        University uni = new University("12345678", "IUST", city);

        // Creating an instance of the Student class and associating it with the
        // University
        Student student = new Student("Farzan", "Rahmani", uni);
    }
}
```

اسکریپت main.py:

```python
from antlr4 import *
from Gen.JavaParser import JavaParser
from Gen.JavaLexer import JavaLexer
from Code.CodeSmellDetectorListener import CodeSmellDetectorListener


def main():
    file_path = "input_with_cycle.java"
    # file_path = "input_without_cycle.java"

    with open(file_path, 'r') as file:
        input_expression = file.read()

    # Create a lexer that feeds off the input expression
    lexer = JavaLexer(InputStream(input_expression))

    # Create a stream of tokens using the lexer
    token_stream = CommonTokenStream(lexer)

    # Create a parser that feeds off the token stream
    parser = JavaParser(token_stream)

    # Obtain the parse tree by invoking the parser's entry point
    parse_tree = parser.compilationUnit()

    # Create a custom listener object
    listener = CodeSmellDetectorListener()

    # Walk the parse tree with the custom listener
```

```
    walker = ParseTreeWalker()
    walker.walk(listener, parse_tree)


# Call the main function
if __name__ == '__main__':
    main()
```

کلاس Listener شخصی سازی شده:

```
from Gen.JavaParser import *
import networkx as nx

class CodeSmellDetectorListener(ParseTreeListener):
    def __init__(self):
        self.graph = nx.DiGraph()

    def addedge(self, edge):
        if isinstance(edge, list):
            self.graph.add_edges_from(edge)
        else:
            self.graph.add_edge(edge)

    def addnode(self, node):
        if not isinstance(node, list):
            if not node in self.graph:
                self.graph.add_node(node)
        else:
            for single_node in node:
                self.addnode(single_node)

    def exitCompilationUnit(self, ctx: JavaParser.CompilationUnitContext):
        cycles = list(nx.simple_cycles(self.graph))
        print(f'The number of cycles detected in the Java code: {len(cycles)}')
        for cycle in cycles:
            print(cycle)

    def enterNormalClassDeclaration(self, ctx:
JavaParser.NormalClassDeclarationContext):
        name = ctx.typeIdentifier().getText()
        self.addnode(name)
        if ctx.classImplements():
            interface_extends = ctx.classImplements().getChild(1)
```

```
            extends = [interface_extends.getChild(i).getText() for i in range(0,
interface_extends.getChildCount(), 2)]
            self.addnode(extends)
            self.addedge([(extend, name) for extend in extends])
        if ctx.classExtends():
            class_extends = ctx.classExtends().getChild(1).getText()
            self.addnode(class_extends)
            self.addedge([(class_extends, name)])

        if ctx.classBody():
            class_body = ctx.classBody()
            for i in range(1, class_body.getChildCount() - 1):
                body = class_body.getChild(i)
                if body.classMemberDeclaration():
                    if body.classMemberDeclaration().fieldDeclaration():
                        if
body.classMemberDeclaration().fieldDeclaration().unannType().unannReferenceType()
:
                            type_var =
body.classMemberDeclaration().fieldDeclaration().unannType().getText()
                            self.addnode(type_var)
                            self.addedge([(name, type_var)])

    def enterNormalInterfaceDeclaration(self, ctx:
JavaParser.NormalInterfaceDeclarationContext):
        name = ctx.typeIdentifier().getText()
        self.addnode(name)
        if ctx.interfaceExtends():
            interface_extends = ctx.interfaceExtends().getChild(1)
            extends = [interface_extends.getChild(i).getText() for i in range(0,
interface_extends.getChildCount(), 2)]
            self.addnode(extends)
            self.addedge([(extend, name) for extend in extends])
```

پایان