گزارش کوتاه از عملکرد بخش‌های مختلف کد

1. راه‌اندازی و اجرای پروژه

فایل grun.bat

این فایل یک اسکریپت دسته‌ای ویندوز است که فرآیند کامل کامپایل و اجرای پروژه را خودکار می‌کند. وظایف اصلی آن :

1. **ایجاد پوشه موقت (temp)** : تمام فایل‌های کامپایل‌شده و تولیدی ANTLR در اینجا قرار می‌گیرند.
2. **اجرای ANTLR** : با استفاده از antlr-4.8-complete.jar، فایل گرامر Language.g4 را پردازش کرده و فایل‌های LanguageLexer.java، LanguageParser.java و LanguageVisitor.java را در پوشه temp تولید می‌کند.
3. **کامپایل** : تمام فایل‌های .java پروژه (شامل فایل‌های سفارشی و فایل‌های تولیدشده توسط ANTLR) را با استفاده از کتابخانه ANTLR کامپایل کرده و فایل‌های .class را در پوشه temp قرار می‌دهد.
4. **اجرا** : کلاس Main را به عنوان نقطه شروع برنامه اجرا می‌کند.

فایل Main.java

این کلاس، نقطه ورود اصلی برنامه است.

- این کلاس در یک حلقه، فایل‌های ورودی را با الگوی Input1.java، Input2.java و... جستجو می‌کند.
- برای هر فایل ورودی که پیدا شود، یک پوشه خروجی همنام (مثلاً Input1) ایجاد می‌کند.
- سپس به ترتیب دو تابع اصلی پروژه را فراخوانی می‌کند:
  1. TokenExtractor.extractTokens(): برای انجام **"بخش اول: استخراج توکن‌ها"**.
  2. SymbolTableBuilder.buildAndPrint(): برای انجام **"بخش دوم: ساخت جدول نماد"**.
- این فرآیند تا زمانی که فایل ورودی بعدی پیدا نشود، ادامه می‌یابد.

2. بخش اول: تحلیل لغوی (Token Extraction)

فایل TokenExtractor.java

این کلاس مسئولیت تحلیل لغوی و دسته‌بندی توکن‌ها را بر عهده دارد.

- **خواندن فایل**: فایل ورودی مشخص‌شده (مثلاً Input1.java) را می‌خواند.
- **استفاده از Lexer**: یک نمونه از LanguageLexer ایجاد می‌کند و تمام توکن‌ها را تا رسیدن به EOF استخراج می‌کند.
- **دسته‌بندی توکن‌ها**: تابع کلیدی getTokenCategory، هر توکن را بر اساس نوع گرامری آن (که از LanguageLexer.VOCABULARY گرفته می‌شود) به 6 دسته اصلی مورد نیاز پروژه تقسیم می‌کند:
  - **Keywords**: با استفاده از یک Set از کلمات کلیدی از پیش تعریف‌شده.
  - **Identifiers**: (شناسه‌ها)
  - **Literals**: (مقادیر ثابت مانند اعداد، رشته‌ها و...)
  - **Operators**: (عملگرها)
  - **Separators**: (جداکننده‌ها)
  - **Comments**: (کامنت‌ها، اگرچه در گرامر Language.g4 کامنت‌ها -> skip شده‌اند، اما منطق آن در اینجا وجود دارد).
- **تولید خروجی**: لیست توکن‌های استخراج‌شده به همراه دسته‌بندی، متن، شماره خط و ستون را در سه محل چاپ می‌کند:
  1. کنسول استاندارد.
  2. فایل متنی (tokens.txt).
  3. یک فایل tokens.html با استایل‌دهی CSS برای نمایش خواناتر در مرورگر.

3. بخش دوم: ساخت جدول نماد (Symbol Table)

این بخش از چندین کلاس مکمل تشکیل شده است که با هم کار می‌کنند تا جدول نماد را بر اساس بازدید از Parse Tree بسازند.

فایل Language.g4

این فایل، تعریف گرامر زبان --Java است. ANTLR از این فایل برای تولید Lexer و Parser استفاده می‌کند. این گرامر ساختار دقیق کلاس‌ها، متدها، فیلدها، عبارات و... را تعریف می‌کند.

فایل Symbol.java

این یک کلاس داده (POJO - Plain Old Java Object) است که برای نگهداری اطلاعات **یک نماد** در جدول نمادها طراحی شده است. این کلاس فیلدهایی برای ذخیره موارد زیر دارد:

- name (نام شناسه)
- type (نوع داده یا نوع بازگشتی)
- kind (نوع نماد: class, interface, method, field, variable, parameter)
- scope (نام حوزه فعلی)
- visibility (سطح دسترسی: public, private, ...)
- parent (نام کلاس والد)
- implementedInterfaces (اینترفیس‌های پیاده‌سازی‌شده)
- isAbstract (آیا abstract است؟)
- isOverride (آیا override است؟)
- parameterList (لیست پارامترها برای متدها)
- initialValue (مقدار اولیه)

فایل Scope.java

این کلاس مسئولیت مدیریت **Nested Scopes** را بر عهده دارد.

- از یک List از Mapها برای نگهداری جداول نماد هر حوزه استفاده می‌کند.
- pushScope (String scopeName): یک حوزه جدید (مانند ورود به یک کلاس یا متد) ایجاد کرده و آن را به بالای پشته اضافه می‌کند.
- popScope: حوزه فعلی را از پشته حذف می‌کند (مانند خروج از متد یا کلاس).
- addSymbol(Symbol sym): نماد جدید را به Map مربوط به **حوزه فعلی** اضافه می‌کند.
- getSymbol(String name): یک شناسه را جستجو می‌کند. این جستجو به درستی از حوزه فعلی شروع شده و در صورت پیدا نشدن، به حوزه‌های بیرونی‌تر (والد) ادامه می‌یابد.

فایل LanguageVisitorImpl.java

این کلاس، برای فرآیند ساخت جدول نماد است.

- این کلاس از LanguageBaseVisitor (تولیدشده توسط ANTLR) ارث‌بری می‌کند و به ما اجازه می‌دهد تا در Parse Tree حرکت کنیم.
- یک نمونه از Scope را در خود نگهداری می‌کند.
- **پیمایش** : با Override کردن متدهای visit... (مانند visitClassDecl ,visitMethodDecl ,visitLocalDecl)، به گره‌های مختلف درخت تجزیه واکنش نشان می‌دهد.
- **ایجاد نماد:**
  - در هر متد visit...، اطلاعات شناسه را از گره فعلی استخراج می‌کند.
  - یک آبجکت Symbol جدید با اطلاعات استخراج‌شده (نام، نوع، سطح دسترسی و...) می‌سازد.
  - آبجکت Symbol ساخته‌شده را با symbolTable.addSymbol() به جدول نماد (در حوزه فعلی) اضافه می‌کند.

فایل SymbolTableBuilder.java

این کلاس فرآیند تحلیل نحوی و ساخت جدول نماد را **Orchestrate** می‌کند.

1. **آماده‌سازی ANTLR**: فایل ورودی را به LanguageLexer و LanguageParser می‌دهد و Parse Tree را با فراخوانی parser.program می‌سازد.
2. **ایجاد Visitor**: یک نمونه از LanguageVisitorImpl می‌سازد.
3. **اجرای Visitor**: متد visitor.visit(tree) را فراخوانی می‌کند. این دستور باعث می‌شود Visitor کل درخت تجزیه را پیمایش کند و در نتیجه، جدول نماد پر شود.
4. **دریافت نتایج**: پس از اتمام پیمایش، جدول نماد کامل‌شده را از visitor.getSymbolTableدریافت می‌کند.
5. **تولید خروجی**: مشابه TokenExtractor، این کلاس تمام نمادهای st.getAllSymbols به‌دست‌آمده را در سه محل چاپ و فرمت‌بندی می‌کند:
   - کنسول.
   - فایل متنی (symbol_table.txt).
   - یک فایل symbol_table.html با استایل‌دهی پیشرفته (مانند رنگ‌بندی بر اساس نوع نماد) برای نمایش خوانای جدول نماد.

ساسان نیک جو
سینا افضلی
محمدحسین طوغانیان