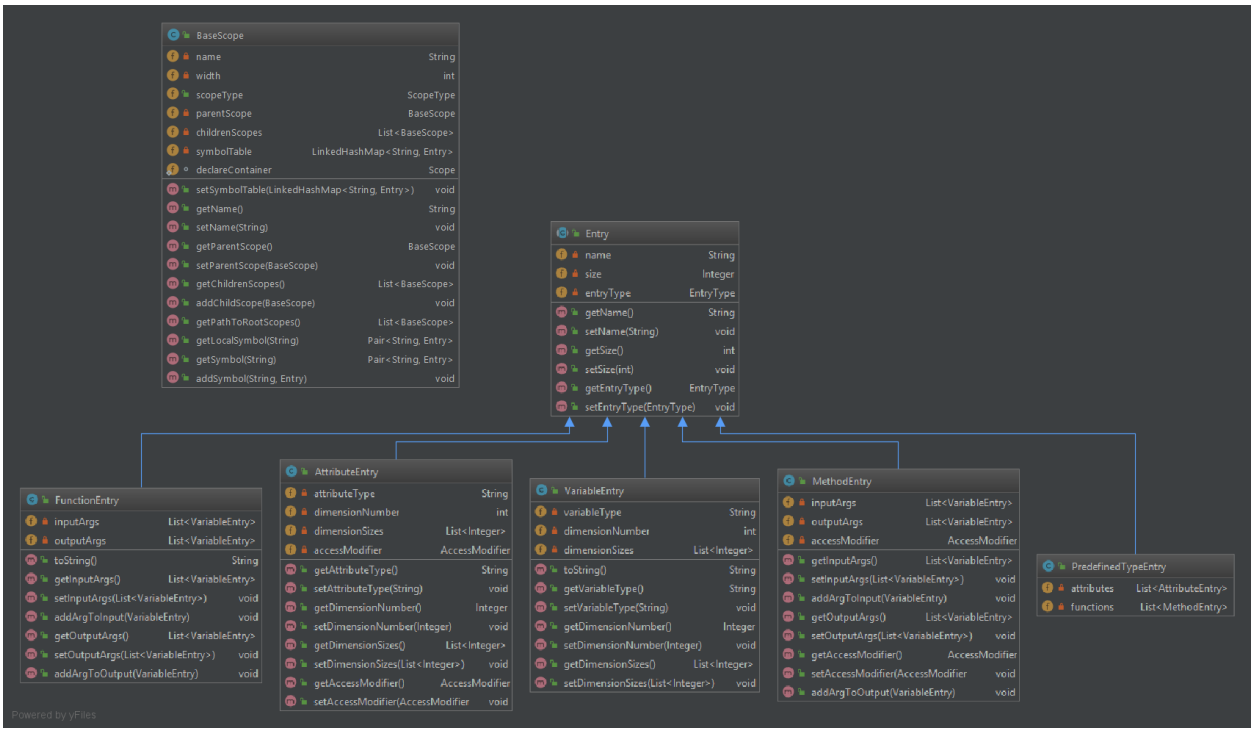
اعضای گروه

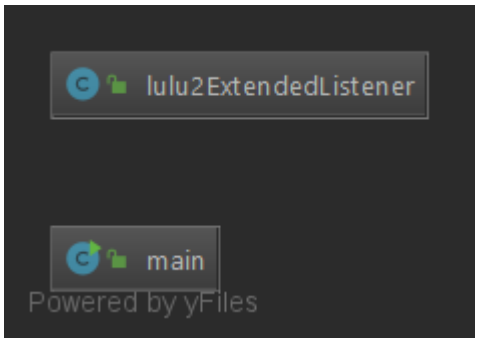
مهدی رفیعی – 953611133034

احمد محمودیان درویشانی – 953611133071

فاز دوم پروژه

پس از مطالعه کتاب ارجاعی antlr reference تصمیم بر آن شد که از میان خروجی‌های antlr که در پوشه gen قرار دارند، از کلاس BaseListener استفاده شود. پس از فکر و بحث در مورد کدهای قابل تولید کلاس بندی پروژه به مانند شکل‌های زیر طراحی شد.





همانطور که در شکل مشخص است، برای ساختن درختی که نشان‌دهنده scopeهای زبان lulu است، از یک کلاس استفاده شده که هر شی از آن دارای شی پدر، اشیا فرزندان و نهایتا یک جدول نشانه می‌باشد. برای پر کردن جدول نشانه از ویژگی abstract class inheritance زبان جاوا استفاده شده، به این ترتیب که جنس جدول نشانه از کلاس Entry است و هر سمبلی که در کد آنالیز می‌شود شی‌ای از یکی از کلاس‌های ارث برده شده از کلاس Entry است. کلاس‌های ارث برده شده بر اساس زبان lulu و قابلیت‌های آن در نظر گرفته شده‌اند.

کلاس lulu2ExtendedListener ارث می‌برد از کلاس lulu2BaseListener که تولیدی antlr است. در این کلاس به ازای هر متغیر غیر پایانی در گرامر دو تابع exit و enter و وجود دارد که با استفاده از کلاس ParseTreeWalker در حین پیموده شدن درخت ساخته شده از String بر اساس گرامر زبان، توابع فراخوانی می‌شوند. پس از ساختن کلاس lulu2ExtendedListener ما به دو تیم تقسیم شده و هر کدام قسمتی جدا از پروژه را به عهده گرفتیم.

احمد محمودیان با در نظر گرفتن شیای staticگونه از جنس scope به نام previous سعی بر این نهاد تا درخت را بسازد. به این ترتیب که به ازای تابع enter گرامرهایی که در آن‌ها scopeای باید ایجاد شود یک scope ساخت و با شی previous که طی الگوریتم همواره شی پدر بود ارتباطات میان درخت را برقرار ساخت. چالش این الگوریتم در گرامرهایی که تو در توی هم بودند نهفته بود.

مهدی رفیعی به منظور آنالیز scopeها برای استخراج جداول نشانه از الگوریتم‌های ریز و درشتی استفاده کرد که با چالش‌های String Manipulation زبان جاوا روبه‌رو بودند. ابتدا در declare scope انالیز شروع شد. ابتدا انالیز متغیرها بود که باید با چرخش روی var_def و var_val به دست می‌امد، که در هر چرخش باید نام متغیر و تعداد بعد آرایه و اعداد داخل آن استخراج میشدند. سپس توابع انالیز شدند که جدای از نام تابع باید متغیرهای خروجی و ورودی را با args و args_var استخراج می‌کرد، که در این مرحله با چالش تشکیل شدن درختی بازگشتی به دلیل استفاده از *args و *args_var روبه‌رو بود. و نهایتا وقتی این مراحل و مراحل ریز دیگری انالیز شدند، سایر نودهای گرامر که ارتباطی با جداول سمبل داشتند مراحلی تقریبا تکراری داشتند، به این ترتیب که صرفا شباهت در انالیزهای وجود داشت و به دلیل یک بار انالیز انجام شده کار بسیار اسان‌تر بود.

نهایتا کدهای مهدی رفیعی و احمد محمودیان با استفاده از BaseScope.addSymbolTable با هم ادغام شد.

ما مهدی رفیعی و احمد محمودیان درویشانی تعهد می‌نماییم که تمام مراحل پروژه تحویل داده شده نتیجه کار گروهی ما بوده و در هیچ یک از بخشهای انحام شده از کار دیگران مپی برداری نشده است. در صورتی که مشخص شود که پروژه تحویل داده شده کار این گروه نبوده است، طبق ظوابط آموزشی با ما برخورد شده و حق اعتراض نخواهیم داشت.