



تمرین TwoStackExpressionEvaluation

در این تمرین می خواهیم الگوریتم دوپشته ای حیاط راه آهن دایجکسترا برای تبدیل یک عبارت میانوند به فرم پسوندی را پیاده سازی نماییم. بسیاری از بخش های این کد از قبل نوشته شده است و دانشجو می تواند بر روی مطالب اصلی درس تمرکز کند.

کلاس اصلی برنامه ExpressionEvaluator نام دارد که دارای دو متود عمومی اصلی زیر است:

```
public String infix2Postfix(String s)
public double evaluateExpression(String s, HashMap<String, Double> variableValues)
```

متود اول رشته متنی یک عبارت ریاضی به فرم میانوند را به رشته متنی فرم پسوندی همان عبارت تبدیل می کند.

متود دوم علاوه بر رشته متنی یک عبارت ریاضی به فرم میانوند، مقدار متغیرهای استفاده شده در آن عبارت را نیز دریافت می کند و پس از تبدیل عبارت به فرم پسوندی مقدار آن را نیز محاسبه می کند.

هرچند متودهای فوق نحوه استفاده از کلاس را مشخص می کنند، اما پیاده سازی آنها نیازمند استفاده از متودهای داخلی دیگر است. به طور مشخص ما ابتدا رشته ورودی را به اجزای تشکیل دهنده آن (Token ها) تجزیه می کنیم و برای هر جزء را در یک کلاس مناسب ذخیره می نماییم. Token می تواند انواع زیر را داشته باشد:

۱- VariableName نوعی از Token است که یک متغیر ریاضی را نشان می دهد.

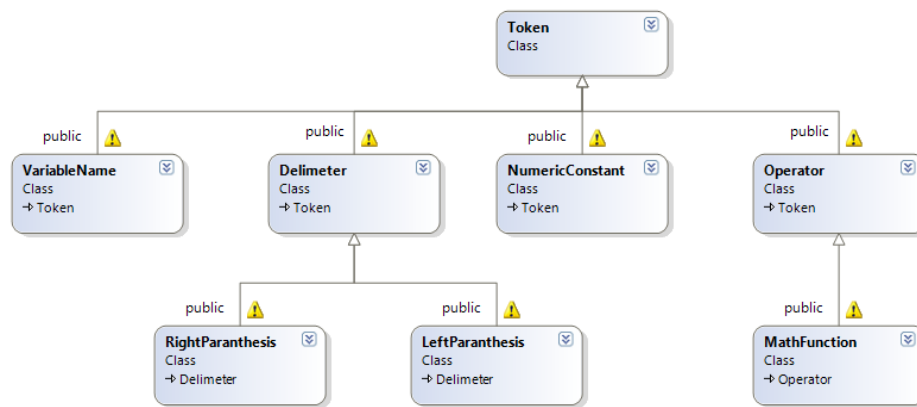
۲- NumericConstant نوعی از Token است که یک ثابت عددی را نشان می دهد.

۳- Operator نوعی از Token است که یک عملگر یگانی یا دوگانی را نشان می دهد.

a. ما توابع (Function) را نیز نوعی عملگر یگانی در نظر می گیریم.

۴- Delimeter شامل پرانتز چپ و راست می باشد.

با در نظر گرفتن هر یک از انواع Token بالا به عنوان یک کلاس و انجام ارث بری مناسب بین آنها به ساختار شکل ۱ می رسمیم (البته در این شکل برخی کلاس ها مانند SinFunc و PlusOperator نشان داده نشده اند).



شکل ۱: دیگرام دسته بندی انواع Token

برای آنکه متودهای عمومی عمل خواسته شده را انجام دهند ما از متودهای داخلی زیر استفاده می کنیم.

```
public void tokenizeString(String s, Vector<Token> out)
```

این متود رشته متنی را به دنباله token های متناظر با آن تجزیه می کند. این متود از قبل نوشته شده است.

```
public String tokenSeq2String(Vector<Token> vt)
```

این متود می تواند دنباله ای از token ها را به صورت یک رشته متنی درآورد. این متود از قبل نوشته شده است.

```
public double evaluatePostfixExpression(Vector<Token> s, HashMap<String, Double> variableValues)
```

این متود با فرض اینکه دنباله token های ارسالی به آن به فرم پسوندی است و با استفاده از نگاشت variableValues که مقدار هر متغیر را می دهد، مقدار عبارت پسوندی را محاسبه می کند. پیاده سازی این متود بخشی از وظایف دانشجویان در این تمرین است.

```
public void infix2Postfix(Vector<Token> infixExpression, Vector<Token> postfixExpression)
```

این متود با فرض اینکه دنباله token های ارسالی به آن به فرم میانوند است، دنباله پسوندی معادل را محاسبه می کند و برمی گرداند. توجه داریم که پارامتر دوم به عنوان خروجی استفاده می شود. پیاده سازی این متود بخشی از وظایف دانشجویان در این تمرین است.

```
public Token getNextToken(StringBuilder ss, Token lastToken)
```

این متود به صورت داخلی توسط متود tokenizeString استفاده می شود.

همچنین در برنامه کلاسی با نام **Operator** وجود دارد که از کلاس **Token** ارث می‌برد. این کلاس در **Constructor** خود ۵ پارامتر دریافت می‌کنند که از چپ عبارتند از:

۱- نام عملگر که از قبل برای بچه‌ها نوشته شده.

۲- اولویت که باید توسط دانشجو تکمیل شود. در این قسمت **Constructor** عددی نسبی دریافت می‌کند که اولویت آن عملگر را نسبت به بقیه مشخص کند. به عنوان مثال اگر به عملگر ضرب عدد ۵ داده شود، به تقسیم که هم اولویت با ضرب است نیز باید ۵ داده شود. اما برای جمع که اولویت پایین‌تری دارد باید عدد کمتری داده شود، مانند ۴. (توجه کنید، اعدادی که در اینجا گفته شده صرفاً مثال بود و لزومی ندارد که این اعداد را به این عملگر‌ها بدهید. بلکه باید با در نظر گرفتن تمام عملگر‌های تعریف شده به همه مقدار مناسبی داده شود)

۳- تجمعی از سمت چپ یا راست که باید توسط دانشجو تکمیل شود. برای مقدار این پارامتر باید یکی از مقادیر **Operator.Associativity.ASSOC_RIGHT** (برای تجمعی از راست) یا **Operator.Associativity.ASSOC_LEFT** (برای تجمعی از چپ) نسبت داده شود. دقت کنید که عملگری مانند جمع تجمعی از سمت چپ است. یعنی برای محاسبه عبارت $2+3+4$ می‌توان تجمعی آن را به صورت $2+(3+4)$ نمایش داد. ولی برای عملگری مانند توان که تجمعی از سمت چپ است عبارت $4^{3^{82}}$ را به صورت $4^{(3^{82})}$ نمایش داد.

۴- مشخص کردن عبارت‌های **Unary** که یک مقدار صحیح یا غلط دریافت می‌کند و از قبل نوشته شده است.

۵- مشخص کننده نوع عبارت که در این برنامه به صورت پیش فرض عملگر یا همان **OPERATOR** و برای توابع نوع آن تابع یا همان **FUNCTION** است. این پارامتر نیز از قبل نوشته شده است.

در کد قسمت‌های ۲ و ۳ که باید توسط دانشجو در کلاس‌های فرزند **Operator** تکمیل شود با سه ستاره (***) جایگزین شده‌اند.

در نهایت به دانشجویان توصیه می‌شود که به تکمیل قسمت‌های ناقص اکتفا نکرده و بقیه کد را نیز مطالعه کنند که بهتر متوجه نحوه عملکرد کد و الگوریتم شوند.

برای انجام این تمرین کارهای زیر را انجام دهید:

- ۱- ابتدا در این پوشه فایل `info.txt` را با مشخصات خود پر کنید.
- ۲- تمرین را انجام دهید و بخش‌های ناقص کد را تکمیل کنید و از درستی پیاده سازی خود مطمئن شوید.
- ۳- همه فایل های اضافی که به دلیل کامپایل برنامه بوجود آمده اند را پاک نمایید.
- ۴- محتویات کل پوشه را به صورت یک فایل `zip` در آورید.
- ۵- مطمئن شوید که وقتی فایل `zip` را باز می کنید پوشه های `src, img` و `test` و همچنین فایل `info.txt` را می بینید.
- ۶- نام این فایل `zip` را به «شماره تمرین-شماره دانشجویی» تغییر دهید (مثل: `123456789- TwoStackExpressionEvaluation.zip`)
- ۷- دقت کنید که پسوند فایل شما حتما `zip` باشد.
- ۸- اگر حجم فایل بالای یک مگابایت باشد، حتما در پاک کردن محتویات بوجود آمده هنگام کامپایل (مرحله ۳) اشتباه کرده اید.
- ۹- ابتدا این فایل را به سیستم «سپهر» ایمیل کنید تا از نحوه عملکرد برنامه خود بر روی تست های تکمیلی آگاه شوید.
- ۱۰- اشکالاتی را که سیستم «سپهر» مشخص کرده است برطرف نمایید و مجددا تمرین را به سیستم «سپهر» تحویل دهید.
- ۱۱- مرحله قبل را آن قدر ادامه دهید که از صحت عملکرد برنامه خود اطمینان حاصل نمایید.
- ۱۲- پس از اطمینان از دریافت نمره کامل، همان فایل ارسالی را از طریق سیستم `VU` تحویل دهید.

با آرزوی موفقیت