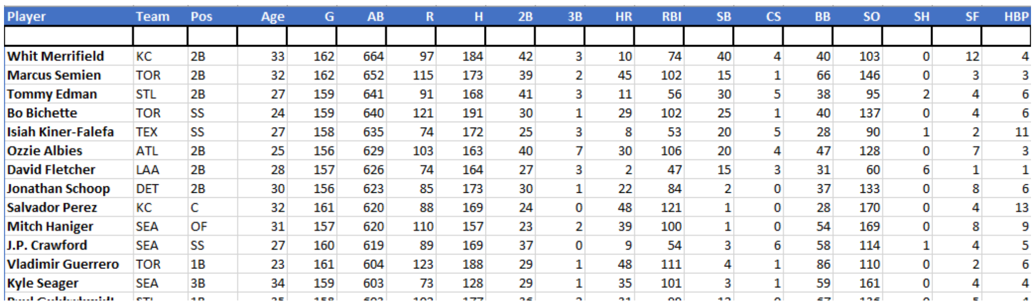
COM-3640 Programming Languages
Homework #2: ANTLR4 User-defined Fields
Due Tue Oct 25 11:59 PM

Background and Motivation

In class we created an expression grammar and used the ANTLR plugin to generate lexer and parser classes. We then used those classes, along with the visitor class that we extended with our logic to implement an evaluate method. This method takes in an expression string as input, evaluates the expression and return the result. Not only did we handle basic expressions such as [*2 + 3*] and [*"one" + "two"*] but we also implemented the ability to set variables and refer to those variables in our expressions – e.g. [*coupon * 2*].

Now we want to use this class to do something useful!

It so happens there is a client who is a big-time fantasy baseball player and has already contracted you to build them a real-time stat viewer. You did a great job. You subscribed to the freely-available data, built them a client web application, and hooked up real-time streaming updates. You gave them the ability to sort and filter on all the available data and they love it. Good job!



Player	Team	Pos	Age	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	SH	SF	HBP
Whit Merrifield	KC	2B	33	162	664	97	184	42	3	10	74	40	4	40	103	0	12	4
Marcus Semien	TOR	2B	32	162	652	115	173	39	2	45	102	15	1	66	146	0	3	3
Tommy Edman	STL	2B	27	159	641	91	168	41	3	11	56	30	5	38	95	2	4	6
Bo Bichette	TOR	SS	24	159	640	121	191	30	1	29	102	25	1	40	137	0	4	6
Isiah Kiner-Falefa	TEX	SS	27	158	635	74	172	25	3	8	53	20	5	28	90	1	2	11
Ozzie Albies	ATL	2B	25	156	629	103	163	40	7	30	106	20	4	47	128	0	7	3
David Fletcher	LAA	2B	28	157	626	74	164	27	3	2	47	15	3	31	60	6	1	1
Jonathan Schoop	DET	2B	30	156	623	85	173	30	1	22	84	2	0	37	133	0	8	6
Salvador Perez	KC	C	32	161	620	88	169	24	0	48	121	1	0	28	170	0	4	13
Mitch Haniger	SEA	OF	31	157	620	110	157	23	2	39	100	1	0	54	169	0	8	9
J.P. Crawford	SEA	SS	27	160	619	89	169	37	0	9	54	3	6	58	114	1	4	5
Vladimir Guerrero	TOR	1B	23	161	604	123	188	29	1	48	111	4	1	86	110	0	2	6
Kyle Seager	SEA	3B	34	159	603	73	128	29	1	35	101	3	1	59	161	0	4	4

You even went further and anticipated that your clients may want to configure which fields are visible and in which order they should be displayed. In your second release, you provided the ability to select and reorder the columns. That was a brilliant move. First, each client can choose to display what he or she wants to see and, second, they don't need to ask you, the programmer, to make the change. Self-service is definitely the way to go.

Your clients are now dependent on your tool and having the data at their fingertips has made them more successful. It doesn't take long, however, for your clients to decide that they need more data in order to make smarter decisions. "What I really need to filter on is the batting average and plate appearances," says one client. "I'm manually calculating the on-base percentage," say another.

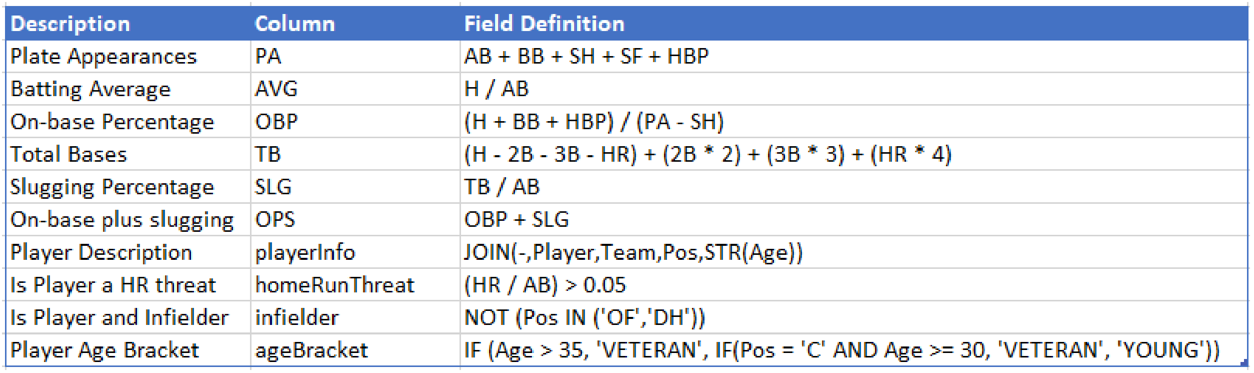
These are simple calculations based on numbers you already have, so you immediately set out to calculate the values and implement 3 more columns. You are almost finished your implementation when a third client emails you and asks if you could include a column for slugging percentage. Similar client emails follow and you quickly realize that this could get out of hand. Moreover, you don't even know how to calculate slugging percentage and some of the other metrics being asked for! The end-of-year technology freeze is around the corner and there is no way you will be able to deliver everything on time. The clients won't be happy if they have to wait a month for the new functionality. Your mind keeps wandering back to a self-service solution …

Suddenly it hits you. Why not let the clients define the new columns for themselves? That way they can each define new columns as it suits them. They will be able to see these columns immediately and not have to wait for you to implement them. You already gave the users the ability to hide or show columns. You can easily extend that control so that the users can create new columns.

As for the column data, since the users have all the underlying information needed for the calculations they want to perform, all you need to do is let them specify an Excel-like formula that defines the calculation. Luckily, the expression language we defined in class handles just such expressions. You decide that you will enhance the language and the code will use the *UserDefinedFields* evaluator method that you created in class to evaluate the columns for each player! Extending the grammar will give your clients the power to define for themselves a whole new range of fields.

Requirements

For this release you will tweak and extend the grammar and visitor class which we implemented to make sure you handle the below new fields for your client users:



Description	Column	Field Definition
Plate Appearances	PA	AB + BB + SH + SF + HBP
Batting Average	AVG	H / AB
On-base Percentage	OBP	(H + BB + HBP) / (PA - SH)
Total Bases	TB	(H - 2B - 3B - HR) + (2B * 2) + (3B * 3) + (HR * 4)
Slugging Percentage	SLG	TB / AB
On-base plus slugging	OPS	OBP + SLG
Player Description	playerInfo	JOIN(-,Player,Team,Pos,STR(Age))
Is Player a HR threat	homeRunThreat	(HR / AB) > 0.05
Is Player and Infielder	infielder	NOT (Pos IN ('OF','DH'))
Player Age Bracket	ageBracket	IF (Age > 35, 'VETERAN', IF(Pos = 'C' AND Age >= 30, 'VETERAN', 'YOUNG'))

You will need to define and implement the following functions and expression.

You should implement all the semantics outlined below, even though not all the functionality is needed for the supplied tests to pass I will be testing, for example, the Boolean <> operator, even though it does not appear in the formulas above.

1) The **STR** function that takes a numerical expression and converts it to a string.
   - Example, `STR(5 + 3) => "8"`

2) The **JOIN** function which:
   - Takes a delimiter character as the first argument and then two or more string arguments.
   - Returns a string, which is a concatenation of all the argument strings with the delimiter in between. (example, `JOIN(-,'A','B','C') => "A-B-C"`)
   - [You need handle only – (minus sign) and / (forward slash) as delimiters]

3) **Boolean expressions**
   - Comparison operators: =, `>`, `>=`, `<`, `<=`, `<>` which operate on both numerical and string expressions (example, `5 > 3`, `'One' <> 'Two'`)
   - The boolean `AND` operator (example, `5 > 3 AND 10 > 3`)
   - The boolean `OR` operator (example, `5 > 3 OR 10 > 3`)
   - AND should have precedence over OR
   - Boolean constants: `TRUE, True, true, FALSE, False, false`

4) The **`IN`** function which tests if a given string matches any of the strings in a set of one or more strings and will return a Boolean value (True if yes, False if no).
   - Example, `'cat' IN ('dog','cat','mouse','fish') => TRUE`
   - Example, `'cat' IN ('dog','mouse','fish') => FALSE`

5) The **`NOT`** function serves as a negation function that takes a Boolean value and outputs the opposite Boolean value (i.e. outputs true if the input is false, and vice-versa).

6) The conditional **IF** expression which takes 3 arguments -- a Boolean expression and two string expressions.
   - If the Boolean expressions is true, then the return value is the result of the first string expression. If the Boolean expression is false, then the result of the second string expressions is returned.
   - Example:

```
IF (5 > 3, 'YES', 'NO') => 'YES'
IF (5 < 3, 'YES', 'NO') => 'NO'
```

Homework Specifics

For this homework we will provide you with a fully-functional Java project which contains the code that we worked on in class (slightly modified).

We are also going to provide you with some sample baseball data and the definitions listed above, as well as working code that reads in the data, processes it and evaluates the fields.

We also hooked all this up to unit tests that you can run straight using maven from the command line or from your IDE.

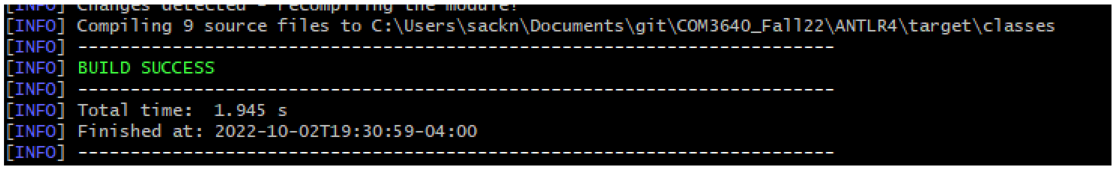
So, what, you ask, is left for you to do??

Your job is to get all the unit tests to succeed!

1) First, download the code from the course repository. This homework's project can be found in the **ANTLR4** folder. https://github.com/Yeshiva-University-CS/COM3640_Fall22

2) You should be able to successfully build the project directly using maven:

```
$ mvn clean compile
```

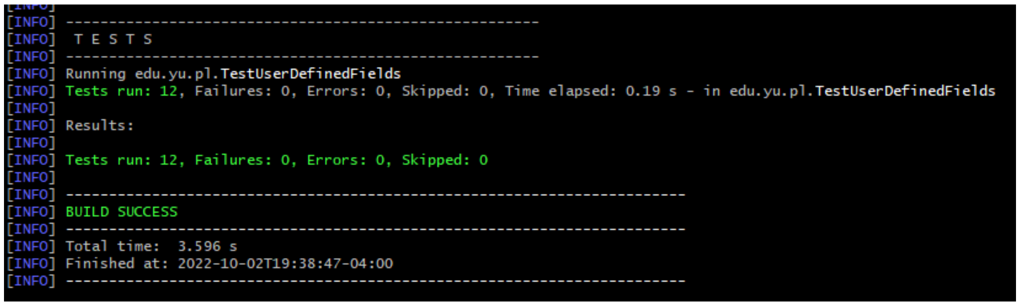


```
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 9 source files to C:\Users\sackn\Documents\git\COM3640_Fall22\ANTLR4\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.945 s
[INFO] Finished at: 2022-10-02T19:30:59-04:00
[INFO] ------------------------------------------------------------------------
```

3) We have divided the unit tests into two files:

i. **TestUserDefinedFields –** This set of unit tests cover what we worked on in class. You should be able to run these tests and they will fully succeed.

```
$ mvn test -Dtest=TestUserDefinedFields
```

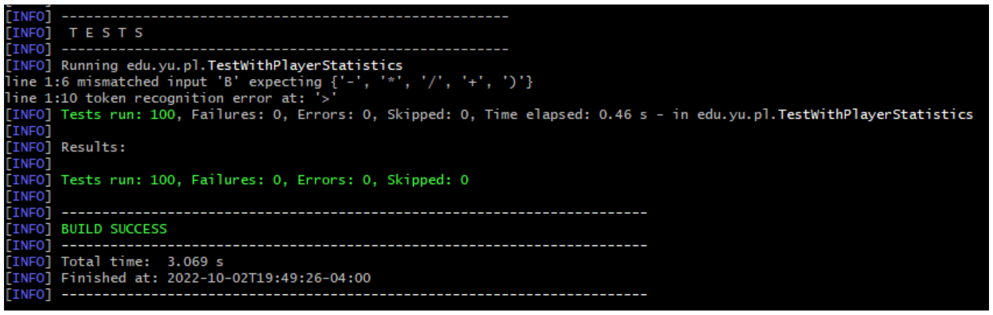


```
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running edu.yu.pl.TestUserDefinedFields
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.19 s - in edu.yu.pl.TestUserDefinedFields
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.596 s
[INFO] Finished at: 2022-10-02T19:38:47-04:00
[INFO] ------------------------------------------------------------------------
```

ii. **TestWithPlayerStatistics –** This file contains tests that load the statistics for 100 players and tests the user-defined formulas listed in the previous section. We have split the formulas into 3 sets and created one test for each set, comparing the results to the actual expected data.

- **Set #1** consists of the first three fields: PA, AVG and OBP. The grammar we wrote in class actually covers these fields and the unit test should complete successfully! (So far you have a perfect score on the assignment ;-)

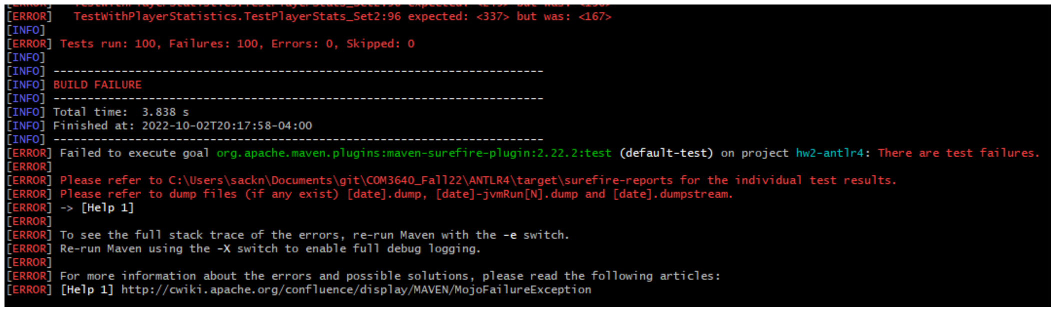
```
$ mvn test -Dtest=TestWithPlayerStatistics#TestPlayerStats_Set1
```



```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running edu.yu.pl.TestWithPlayerStatistics
line 1:6 mismatched input 'B' expecting {'-', '*', '/', '+', ')'}
line 1:10 token recognition error at: '>'
[INFO] Tests run: 100, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.46 s - in edu.yu.pl.TestWithPlayerStatistics
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 100, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.069 s
[INFO] Finished at: 2022-10-02T19:49:26-04:00
[INFO] ------------------------------------------------------------------------
```

- **Set #2** consists of the second three fields: TB, SLG and OPS. Running these tests should fail.

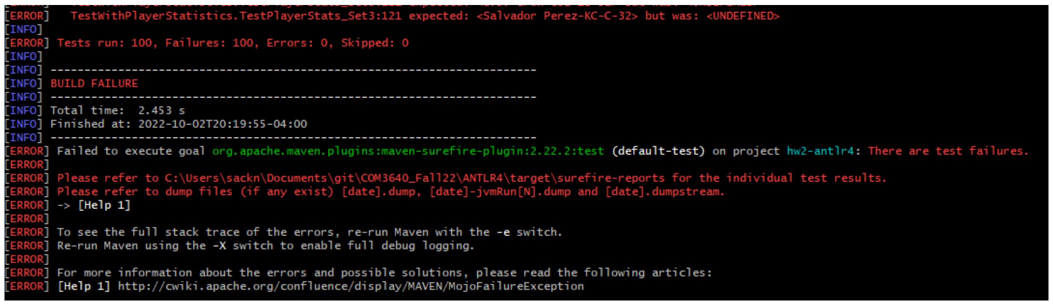
```
$ mvn test -Dtest=TestWithPlayerStatistics#TestPlayerStats_Set2
```



```
[ERROR]   TestWithPlayerStatistics.TestPlayerStats_Set2:96 expected: <337> but was: <167>
[INFO]
[ERROR] Tests run: 100, Failures: 100, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.838 s
[INFO] Finished at: 2022-10-02T20:17:58-04:00
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.2:test (default-test) on project hw2-antlr4: There are test failures.
[ERROR]
[ERROR] Please refer to C:\Users\sackn\Documents\git\COM3640_Fall22\ANTLR4\target\surefire-reports for the individual test results.
[ERROR] Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
```

- **Set #3** consists of the last four fields. Running these tests should also fail.

```
$ mvn test -Dtest=TestWithPlayerStatistics#TestPlayerStats_Set3
```



```
[ERROR]   TestWithPlayerStatistics.TestPlayerStats_Set3:121 expected: <Salvador Perez-KC-C-32> but was: <UNDEFINED>
[INFO]
[ERROR] Tests run: 100, Failures: 100, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.453 s
[INFO] Finished at: 2022-10-02T20:19:55-04:00
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.2:test (default-test) on project hw2-antlr4: There are test failures.
[ERROR]
[ERROR] Please refer to C:\Users\sackn\Documents\git\COM3640_Fall22\ANTLR4\target\surefire-reports for the individual test results.
[ERROR] Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date].dumpstream.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
```

Your task is to tweak the grammar and extend it and the handlers according to the requirements so that all the unit tests succeed. **Note there are requirement cases that are not covered by the 10 new client fields and, hence, the unit tests. Please make sure to read the requirements carefully so that you implement all the functionally required.**

(Note, you should also be able to load the project into Intellij and build from there. As we saw in class, after compiling the first time, you may need to right-click on the target/generated-sources/antlr4 directory within the project folder and mark the directory as "Generated Sources Root.")

General HW Submission Instructions

- All submitted assignments must be checked into the YU GitHub system.

- Say that your Git url is https://github.com/Yeshiva-University-CS/SmithBob. When you *git clone* that url to your computer, the result is a directory named SmithBob. That root (Git) directory will be referred to as **$MYGIT**.

- All of your submitted assignments must be rooted in a directory named: **$MYGIT/PL-COM3640/assignments**.

- Each assignment will be associated with a specific subdirectory. We will refer to that directory name as **$DIR.**

- Therefore, all submissions for a given assignment will be checked into and rooted at:
  $MYGIT/PL-COM3640/assignments/$DIR

This HW's Submission

- The subdirectory for this assignment is **ANTLR4**.
- The assignment must be checked into: **$MYGIT/PL-COM3640/assignments/ANTLR4**