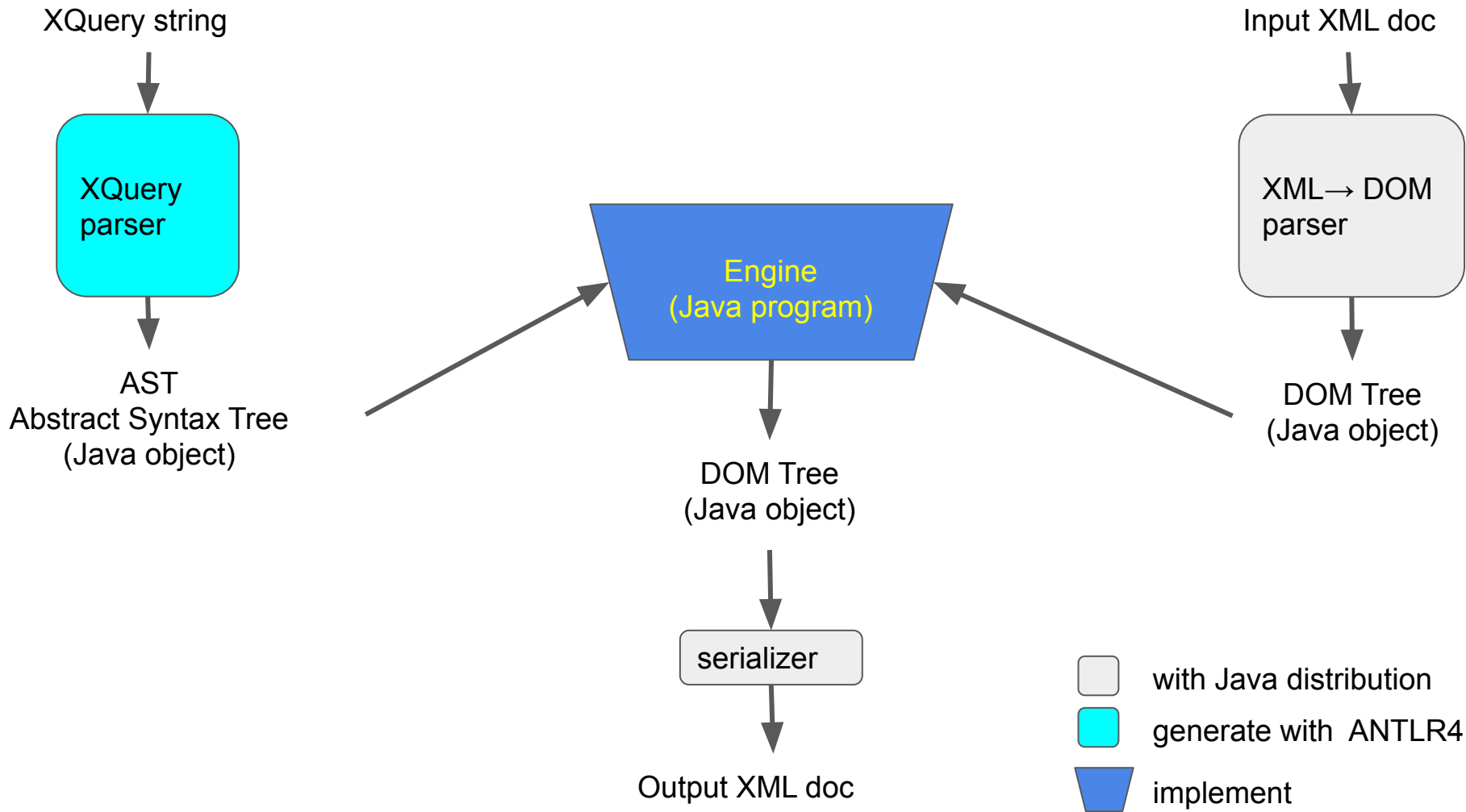



# CSE 232B

## Project Architecture



# The XPath/XQuery Parser

A cyan rounded square icon with a thin black border. Inside the square, the text "XQuery parser" is written in black, with "XQuery" on the top line and "parser" on the bottom line.

XQuery  
parser

Generated by you from the grammar given, using ANTLR 4.

ANTLR 4 takes as input a grammar  $G$  and outputs the Java code of a parser  $P$ .

When called with an input expression  $E$ , the parser  $P$  generates an AST (Abstract Syntax Tree) of  $E$  as a Java object.

# Abstract Syntax Trees (ASTs): Rules

Reflects the structure of an expression according to the grammar rules.

Example: Grammar production rules:

$e \rightarrow \text{INT}$	#ExprInt
$  (e)$	#ExprParen
$  e * e$	#ExprMult
$  e / e$	#ExprDiv
$  e + e$	#ExprAdd
$  e - e$	#ExprSub

Lexer/tokenizer rules:      INT defined by regex  $[1-9][0-9]^*$

# Abstract Syntax Trees (ASTs): Parsing

Input Expression:  $13 + (3 * 5)$

Tokenized as:  $\text{INT} + ( \text{INT} * \text{INT} )$

Parsed as:  $13 + (3*5)$

$\Rightarrow \text{INT} + ( \text{INT} * \text{INT} )$  by tokenizer rule

$\Rightarrow e + ( e * e )$  by rule ExprInt

$\Rightarrow e + (e)$  by rule ExprMult

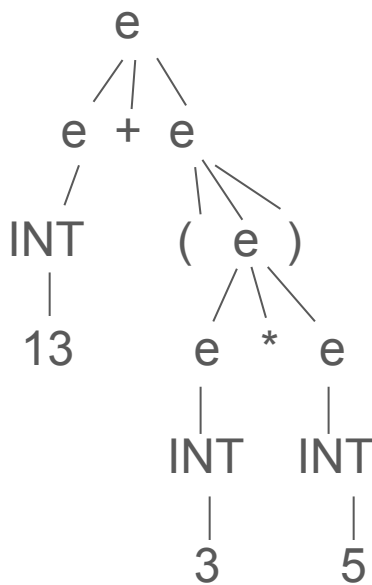
$\Rightarrow e + e$  by rule ExprParen

$\Rightarrow e$  by rule ExprAdd

# AST

$13 + (3 * 5) \Rightarrow \text{INT} + ( \text{INT} * \text{INT} ) \Rightarrow e + ( e * e ) \Rightarrow e + (e) \Rightarrow e + e \Rightarrow e$

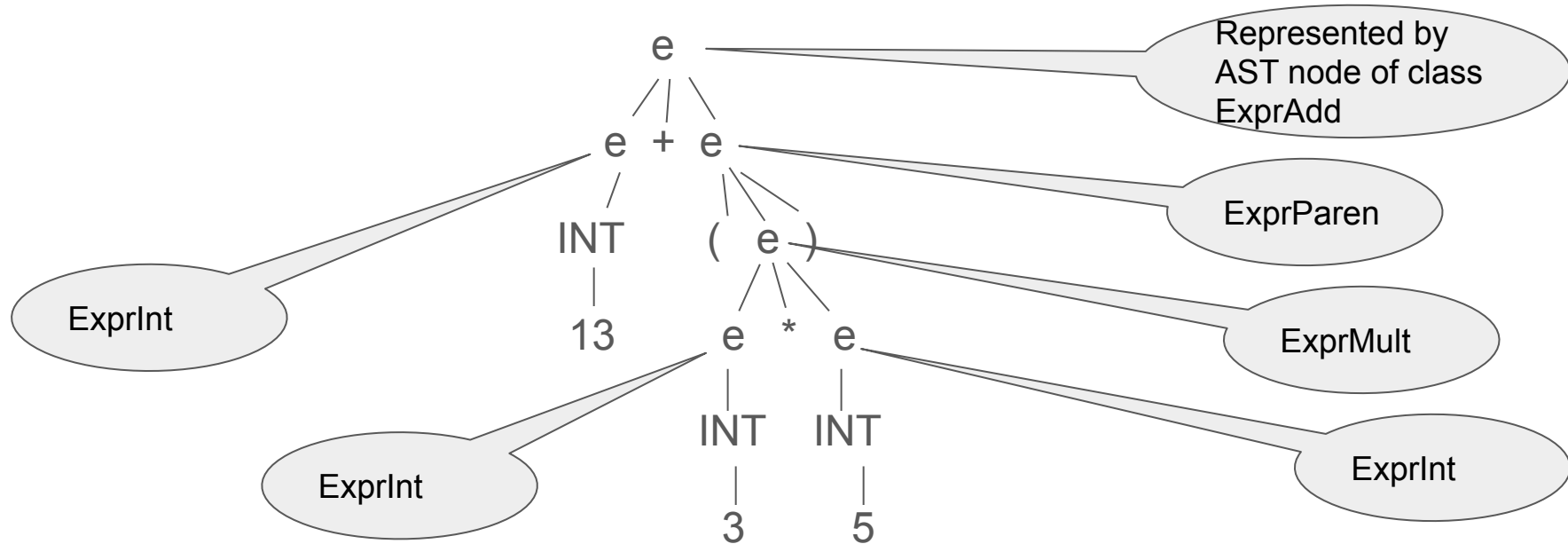
AST reflects parsing sequence: for each rule application  $p \Rightarrow c$ ,  $p$  is parent of  $c$



# AST represented in ANTRL 4

$13 + (3 * 5) \Rightarrow \text{INT} + ( \text{INT} * \text{INT} ) \Rightarrow e + ( e * e ) \Rightarrow e + (e) \Rightarrow e + e \Rightarrow e$

AST reflects parsing sequence: for each rule application  $p \Rightarrow c$ ,  $p$  is parent of  $c$



# The Internal Representation of the XML Data

XML-->DOM  
parser

The XML data will be represented as a Java object that implements a DOM tree.

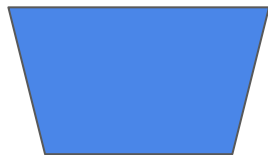
Recall, DOM (Document Object Model) is a set of interfaces standardized by the World-Wide Web Consortium (W3C).

The Java distribution comes with an XML→ DOM parser.

*You do not have to implement this module, just use the existing one.*



# The Evaluation Engine



The evaluation engine takes as input

- An AST representing the query  $Q$
- A DOM tree representing the XML data  $X$

and it outputs a DOM tree representing the result of  $Q$  on  $X$ .

**This is what you implement**, using the semantics specification as pseudo-code. The pseudo-code already implementing the recursive evaluation functions.

Your task is to to translate the pseudocode to JAVa.

# Serializing the DOM Output to XML

serializer

The DOM tree produced by your engine will be output in XML-serialized form.

*You do not have to implement this functionality, just use the one provided by the DOM implementation that comes with the Java distribution.*