Use the ***Interpreter Pattern*** to build an interpreter for a language.

***Scenario – Duck Simulator*** – Each child gets to control one duck with a simple language. Sample language,

***right; while (daylight) fly; quack;***

When you need to implement a simple language, the Interpreter Pattern defines a class-based representation for its grammar along with an interpreter to interpret its sentences. To represent the language, you use a class to represent each rule in the language. To interpret the language, call the interpret() method on each expression type. This method is passed a context-which contains the input stream of the program we are parsing and matches the input and evaluates it.

Sequence

expression1

Expression2

Interpret(context)

Repetition

variable

expression

interpret(context)

FlyCommand

interpret(context)

RightCommand

interpret(context)

QuackCommand

interpret(context)

Variable

interpret(context)

Expression

interpret(context)

***Benefits***:

* Representing each grammar rule in a class makes the language easy to implement.
* Because the grammar is represented by classes, you can easily change or extend the language.
* By adding methods to the class structure, you can add new behaviors beyond interpretation, like pretty printing and more sophisticated program validation.

***Uses and Drawbacks:***

* Use interpreter when you need to implement a simple language.
* Appropriate when you have a simple grammar and simplicity is more important than efficiency.
* Used for scripting and programming languages.
* This pattern can become cumbersome when the number of grammar rules is large. In these cases a parser/compiler generator may be more appropriate.