Object Oriented Analysis and Design with Java

UE19CS353

Prof. Spurthi N Anjan
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

PES UNIVERSITY ONLINE

UE19CS353: Object Oriented Analysis and Design with Java

OO Design Patterns & Anti-Patterns with Sample implementation in Java

Department of Computer Science and Engineering

UE19CS353: Object Oriented Analysis and Design with Java

Unit-5

Behavioral Patterns – Interpreter

Department of Computer Science and Engineering

Interpreter

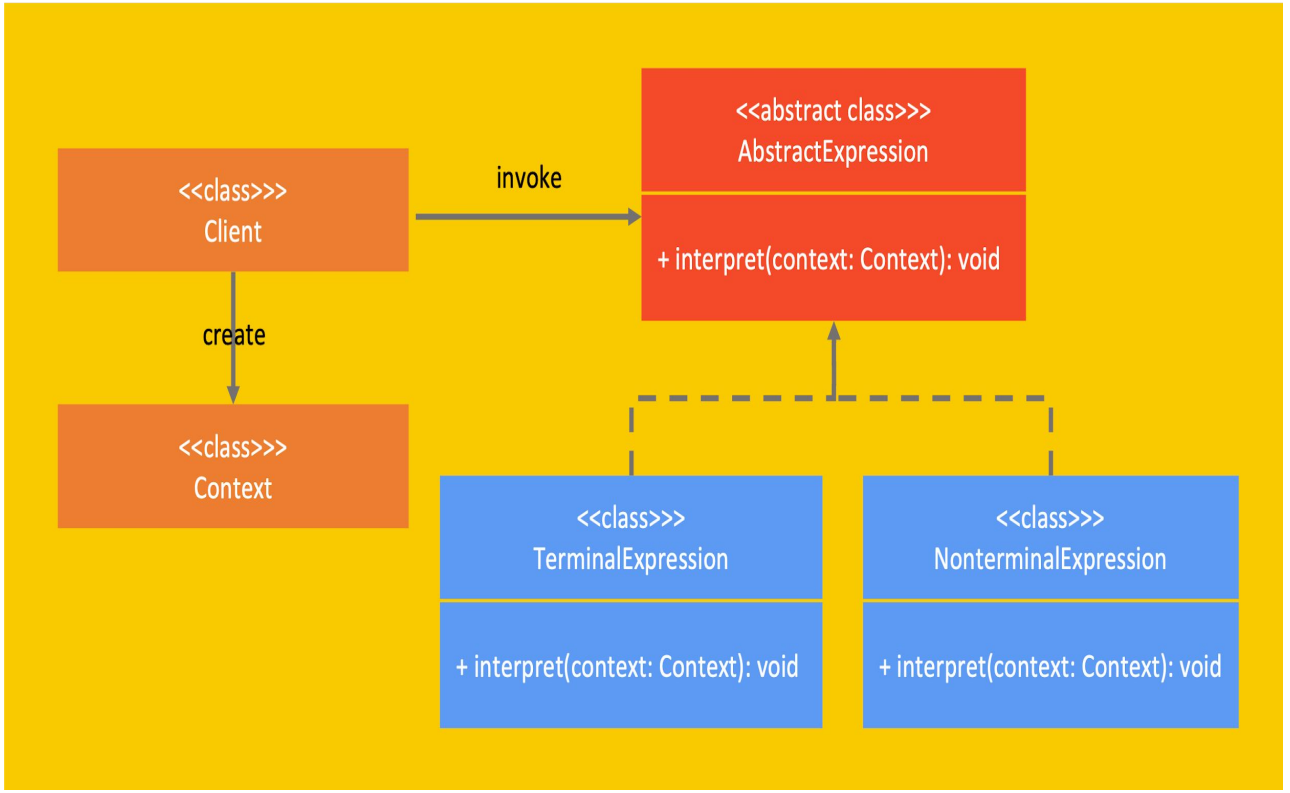
- Interpreter pattern provides a way to evaluate language grammar or expression. This type of pattern comes under behavioral pattern.
- This pattern involves implementing an expression interface which tells to interpret a particular context.
- The Interpreter pattern is used to define a grammatical representation for a language and also provides an interpreter to deal with the grammar.
- The basic idea of the Interpreter pattern is to have a class for each symbol:
  - **Terminal** - are the elementary symbols of the language defined by a formal grammar.
  - **NonTerminal** - also known as syntactic variables that are replaced by groups of terminal symbols according to the production rules. The NonTerminal uses a composite design pattern in general.

Object Oriented Analysis and Design with Java

Interpreter

Component classes in the diagram include:

- Client: is a class that uses the interpreter design pattern
- Context: is the class created by the client and passed to the interpreter, it contains input data and output results.
- AbstractExpression: is the base class containing the interpret function, the Client will use this class instead of the concrete implementation classes
- TerminalExpression: is a class that inherits AbstractExpression and implements the interpret function, in fact there will be many TerminalExpression classes
- NonterminalExpression: is a class that inherits AbstractExpression, but it will have many subclasses and call these subclasses in the interpret setting function.



Interpreter

Interpreter implements the following goals:

- Create a model where the user only needs to care about the input data and receive the output data.
- Create classes that implement interpret function, reduce if else forked statements, thereby increasing extensibility later

Interpreter Example

Let's take an example of an algebraic expression 2 + 3. To model this expression we need ConstantExpression that represents the constant operands(2 and 3) and then AndExpression that takes two ConstantExpression expressions operands and performs addition operation. All Expression classes must implement a common interface 'Expression'.

Above expression can Modelled as :

AndExpression(ConstantExpression(2), ConstantExpression(3))

Similarly, (2 + 6) + (1 + 9) can be modelled as:

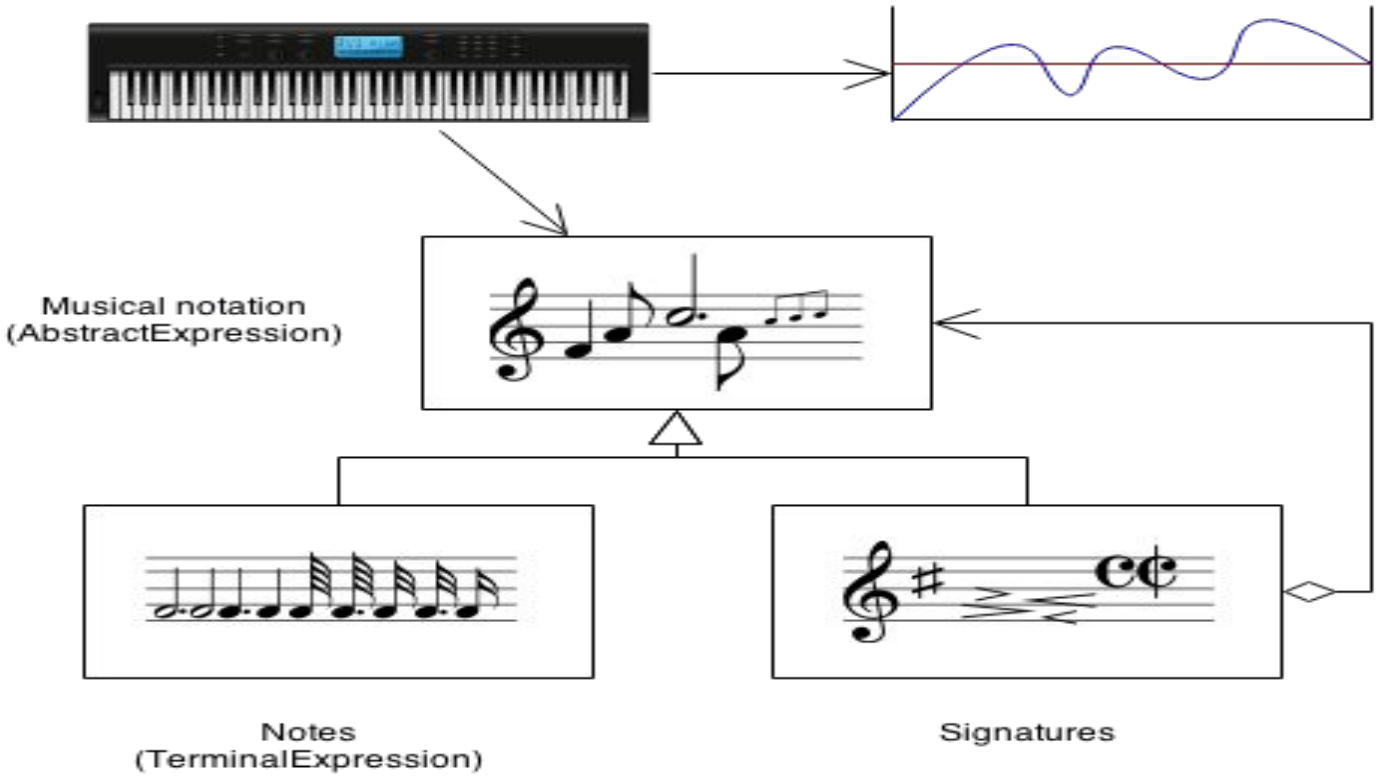
AndExpression(AndExpression(ConstantExpression(2), ConstantExpression(6)),

AndExpression(ConstantExpression(1), ConstantExpression(9)))

Interpreter Example

The Interpreter pattern defines a grammatical representation for a language and an interpreter to interpret the grammar.

Musicians are examples of Interpreters. The pitch of a sound and its duration can be represented in musical notation on a staff. This notation provides the language of music. Musicians playing the music from the score are able to reproduce the original pitch and duration of each sound represented.



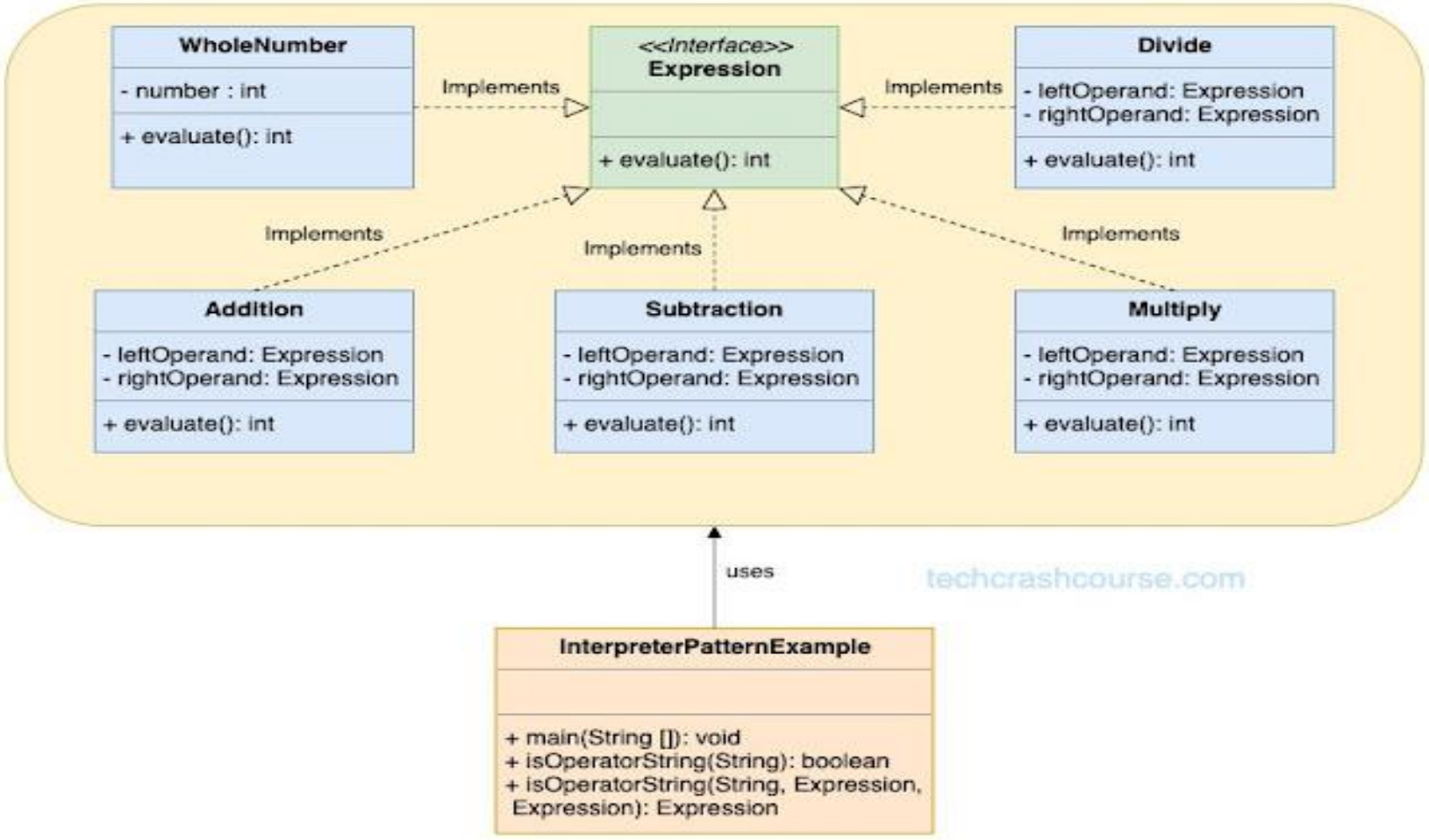
Advantages of Interpreter Pattern

- It defines a class bases hierarchical representation of language grammar.
- We can represent each language grammar rules(semantics) as a class implementing a common evaluate interface. It becomes easy to implement the language because parser treats each rule identically.
- It helps in segregating the ownership of a each grammar rule to a particular class.
- Each expression class hides it's specific evaluation logic from language parser.
- We can extend the language by adding more expressions or change the evaluation logic of any expression without modifying language parser code.

When we should use Interpreter Pattern

- It is not suitable for building a whole large interpreter for a language. It can be used when the grammar of the language is simple.
- When we want to model a simple a recursive grammar.
- When simple implementation is high priority then than efficiency.

Interpreter



Use cases of Interpreter Design Pattern

Programming language compilers, interpreters, IDEs, Document readers like HTML, XML, PDF, etc. A regular expression is a very subtle example of Interpreter.

Interpreter Design Pattern in JDK

- util.Pattern
- text.Normalizer
- text.Format

Interpreter Link

https://www.techcrashcourse.com/2015/10/interpreter-design-pattern-in-java.html

THANK YOU

Prof. Spurthi N Anjan

Department of Computer Science and Engineering