Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 3159-0040
Vol. 2 Issue 3, March - 2015

Comparison between any Two Design Patterns

Tarek Eltaieb
Professor, Department of CS, University of Bridgeport
Bridgeport, Connecticut.
taltaeib@my.bridgeport.edu

Mamatha Rithwika Gaddam
Student, Department of CS, University of Bridgeport
Bridgeport, Connecticut
mgaddam@my.bridgeport.edu

***Abstract*—Design patterns are the solutions to any software design problems which we find in real world application development. These design patterns show typical relationships and interactions between classes and objects.**

Introduction

A pattern is a way to define a solution or approach to a common design problem while coding or implementing the code into an application. In object oriented design pattern, relation between classes and objects. A pattern has important aspects; they are name of pattern, Problems in the pattern, Solutions to the problem and Consequences for the problem. For any design pattern these are very important. We are focus on comparing two or three design patterns available for research, namely

- Creational patterns
- Behavioral patterns
- Structural patterns

Uses of Design Patterns

Design patterns can build up the process of development by giving the design tested and improved programs. For the software design to become effective we need to consider some issues, this issues may affect the design at the time of implementation but not before implementation. So the design issues are important, they can cause major problems. Design patterns are very helpful to prevent the issues and this also enhances code readability.

Few designers know how to apply specific software design techniques to specific issues. But these designs are difficult to apply to a wide range of design issues. Design patterns give the solutions, in a documented format which does not require specific issues to a particular problem.

Creational Pattern:

Possible Issue:

The application should be changed into platform dependent like database, windows, operating systems, so that the application can be transported to other system from one system. This object oriented programming is not well designed and build.

Creational patterns	Behavioral patterns
Class instantiation is the main aspect of creational patterns. There are two types class and object creational patterns.	Communication between objects is the main aspect of behavioral patterns.
Factory method: An instance is created for different hierarchy of classes.	Interpreter: A way to define elements of language.
Abstract Factory method: Same as factory method.	Chain of responsibility: Request should be passed between set of objects by calling methods.
Builder: The object formed from its representation is separated.	Command: Develop a command request in any form of object.
Object pool: Expensive investment and production should be considered and removed by recycling the objects.	Iterator: Elements of library should be accessed one by one.
Prototype: When an instance is designed and defined then it should be performed.	Mediator: There is a communication between class and object.
Singleton: Single instance can subsist in a method of library class.	Memento: An objects internal state can be captured and built again.

Discussion:

To abstract creating the order of related or dependent objects without knowing the classes, it provides the level of indirection. To provide the services for the platform, the "factory" object is responsible. But these clients won't create the platform for factory objects; they just replace the objects from other platforms.

Because of these replacing of products is easy, as the factory object of particular class arrive once in the entire application. Class instantiation can be done in

this case by replacing it with different instance of the abstract factory.

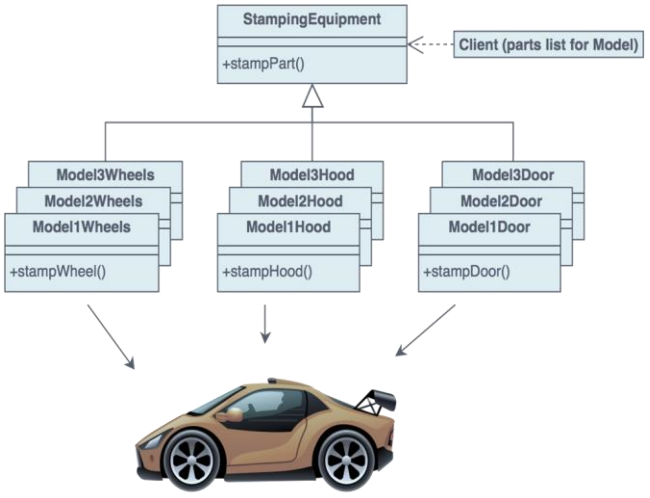
It is implemented as Singleton method because the services which are provided by factory object are so prevalent.

Structure of design:

The factory methods are designed for every product in the abstract factory design pattern. The new product classes which are platform specific are summarized by the methods. Then the factory derived class defines its own platform.

Implementation of method:

Providing an interface for creating tree like structure for all related class object without mentioning concrete class, this is the main purpose of Abstract factory. The best example for this pattern is Sheet metal stamping device which is used for making automobiles. The stamping device which creates body parts acts as an abstract factory. To stamp the parts of automobiles like Right side doors, Left side doors, Right and left vendors, etc. use the same Factory method. By using the rollers to change the stamp, concrete classes which are produced by the machinery should be used.



Check list:

1. The source for the pattern should be platform independent and their services should be defined and decided.
2. Platforms and products should be matched in any design pattern and they should be represented in matrix form.
3. Methods should be created by interfaces per code.

Behavioral Pattern:

Aim:

This pattern is used to represent grammar using an interpreter, which interprets a sentence of the grammar in any given language. A map should be plotted from language to grammar to the design. So that it is easy to interpret the grammar in the language.

Possible issue:

In any language, the errors are common in the grammar, so the languages should be designed correctly so that no problems occur in the know language. Many problems can be resolved with the interpretation engine.

Discussion:

The interpreter design includes three steps:

1. Designing and defining a domain language (i.e. characterization of problem) as easy and understandable language.

2. Defining the rules for a language for formation of sentence.

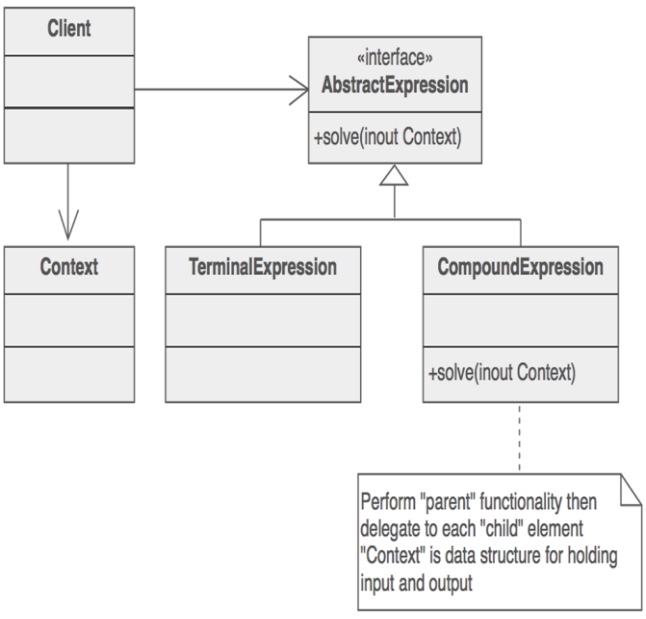
3. After forming sentences, interpreting these sentences to solve different design issues for particular language.

The grammar rules can be represented by patterns which are using classes from the order. As these grammars are usually hierarchical structure, classes map correctly because of inheritance.

Abstract base class uses the interpret method and the subclasses use the before class interpret method along with the current state of class in language and adding them in solving the problem issue. Interpret () method is used in this design pattern of any language.

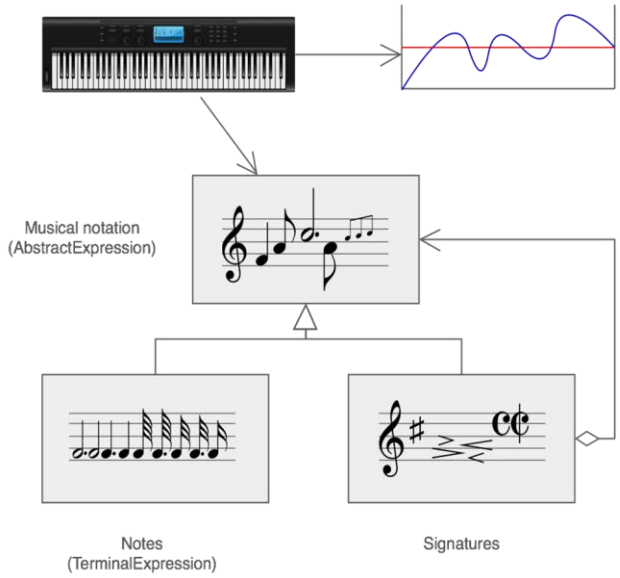
Structure of design:

In the interpreter design pattern, any grammar is designed using few particular rules for particular language. Every rule in the grammar is of two types, one is compound and other is terminal which is leaf node. For any sentence to form first the context is needed which explains the grammar of language through recursive traversal.



Implementation:

The Interpreter pattern is defined as grammatical representation for any language and an interpreter to change the rules to form sentences. Example of interpreter pattern is Musicians. To create any music first the notes and symbols are very important. The musicians first have to know about the music notes. Then they start playing the music using these notes on the score, so the music played from it will reproduce the pitch and also the duration of each note of the sound. Then the final graph is represented and duration is noted.



Check list:

1. First we have to check whether the language will offer agreed return on investment.
2. Grammar is most important for any language, it should be defined.
3. Each production should be mapped to a class in the grammar.
4. After mapping all classes, these all should be arranged into the structure of the pattern.
5. In the hierarchy tree structure context should be defined also called as interpret () method.
6. In a language, the base node is defined and output is obtained, then this output is combined with the current state input and finally added to the context object for processing.

Conclusion:

The common problems can be solved using Design patterns. The design principles should be understood correctly, so that we can know where to use particular designs. For a good design, identifying design objectives is very important. They can be changed by time and make them more robust design pattern methods.

References

Books:

1) Design Patterns Explained simply.

2) Modern c++ design

3) Applying UML and Patterns.

Web:

1) Random Article on Japanese automobile design.

2) www.oodesign.com

3) www.pluralsight.com