Design Pattern

Table of Contents

[Creational Design Pattern 1](#_Toc499448861)

[Structural Design Pattern 1](#_Toc499448862)

[Strategy Design pattern 1](#_Toc499448863)

[Command Pattern 2](#_Toc499448864)

[Behavior Design Pattern 2](#_Toc499448865)

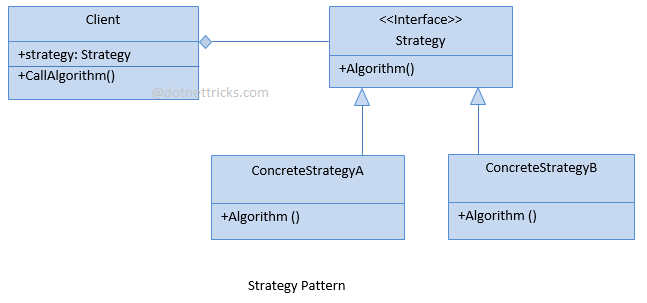
# Creational Design Pattern

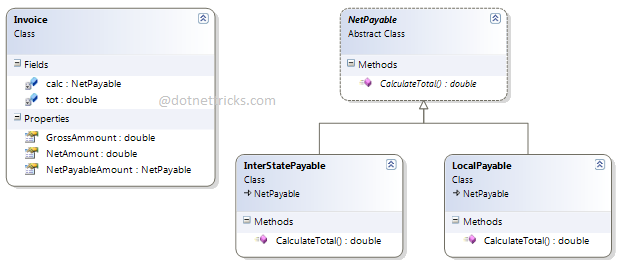
# Structural Design Pattern

## Strategy Design pattern

This pattern allows a client to choose an algorithm from a family of algorithms at run-time and gives it a simple way to access it.

This pattern involves the removal of an algorithm from its host class and putting it in a separate class. As you know, there may be multiple strategies which are applicable for a given problem. So, if the algorithms will exist in the host class, then it will result a messy code with lots of conditional statements.

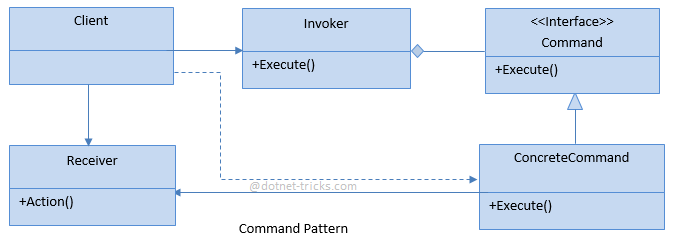




## Command Pattern

In this pattern, a request is wrapped under an object as a command and passed to invoker object. Invoker object pass the command to the appropriate object which can handle it and that object executes the command. This handles the request in traditional ways like as queuing and callbacks.

This pattern is commonly used in the menu systems of many applications such as Editor, IDE etc.



# Behavior Design Pattern