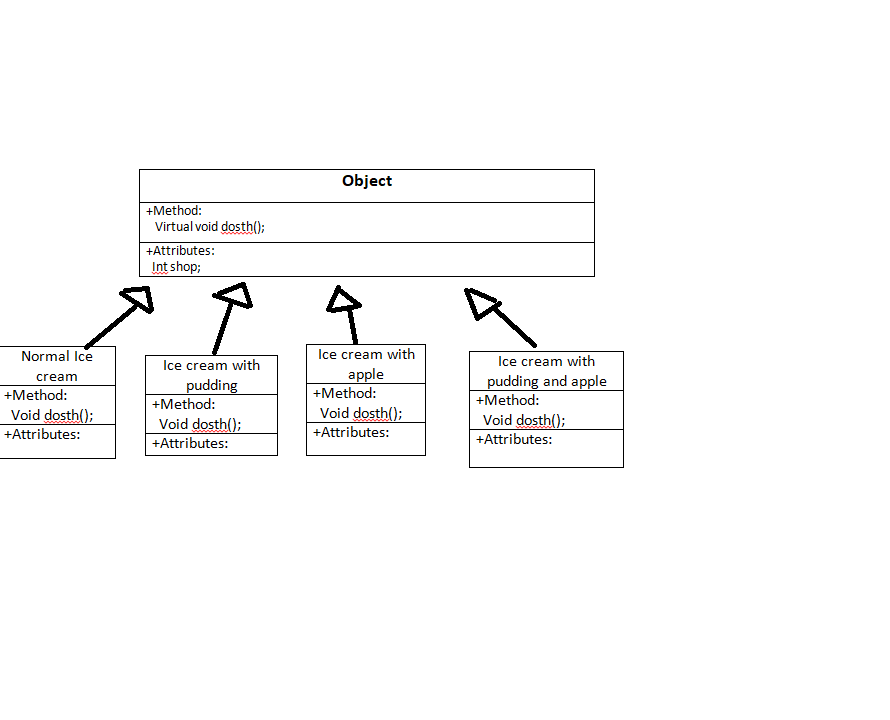
**DECORATOR PATTERN**

**1.Problem:**

-Suppose you are planning to open an ice cream restaurant and you seriously want to earn a lot of money from this restaurant. Let’s say you have many kind of ice cream in your menu and the basic one is a normal ice cream. Customers can add any kind of their favorite ingredients such as: pudding , apple , pineapple ,… Let’s use object oriented programming to solve this problem.

**2.Proposed method without using pattern:**

-First step: you declare an abstract base class Object.

-Second step: When customers want to order for their favorite dishes, you have to implement such a new kind of class depending on the demand of customers . This new class will inherit from the base class Object and it has its own attributes about the description of the customers’ demand .This process is repeated every time the customer request a new kind of dishes . Class diagram of this method is shown below:

Let’s take a look at source code implementing in C++ in Normal Method folder for more detail.

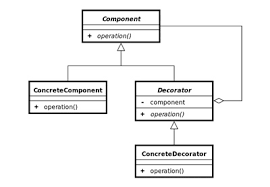
So the above method is not good when you really want to make money from this start up project because it wastes you a lot of resources and money for a new kind of ice cream .Instead of that you can take advantage of the available normal ice cream so that every time customer want to add any kind of ingredients, you just normally add it to this normal ice cream and don’t have to create a new kind of ice cream called “pudding ice cream” or “apple ice cream”. So how can you transform this idea into implementing? This is when decorator pattern can be used to optimize your resources and money. Let’s see what decorator pattern is and how we can use it!!

**3.Introduction to decorator pattern:**

-Decorator is used when you want to add a new functionality to an existing object without changing its structure .This type of pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

-Decorator pattern will create a new abstract class called Decorator Abstract class which is inherited from the base class Object and it also ‘has a’ pointer to Object member which points to a specific Object for wrapping it .

-Every time to want to create a new decorator class for wrapping the existing object, just initialize a new class which is inherited from the Decorator Abstract class . Class diagram for a standard implementation of Decorator Pattern:



+Component class is the Object class

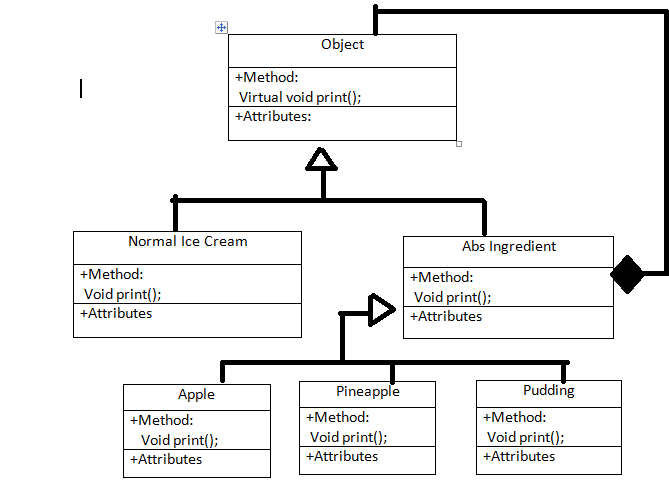
+Concrete Component class is an existing object which you want to add functionality

+Decorator class is Decorator Abstract Class which is the main key of this kind of pattern

+Concrete Decorator is a kind of functionality you want to wrap for existing object

**4.Let’s use this pattern for solving our problem:**

+Class diagram of decorator pattern for solving ice cream restaurant problem:



-Every time the customer wants to add a new ingredient to an existing ice cream , you just need to call the ingredient of that object for wrapping it with that ingredient .For example:

+If the customer1 wants to buy a normal ice cream with apple ingredient . You can easily do it in one step: apple(normal ice cream)

+Customer2 wants to buy a normal ice cream with apple and pudding ingredient. Do the same things as before : apple(pudding(normal ice cream))