Design Pattern : Observer, Strategy, Visitor

Bonjour à tous et bienvenue dans ce workshop d'introduction aux patrons de conception (ou design pattern) observateur, stratégie et visiteur. Nous allons pas à pas expliquer l'intérêt de ces modèles et travailler sur leur compréhension et leur implémentation en C++.

Pour vous éviter de réécrire toutes les classes, je vous mets à disposition un repository avec tout le code déjà écrit. Il ne manque que le code que vous allez écrire durant ce workshop. https://github.com/Preant/design-pattern-workshop

Observateur :

Ce design pattern de la catégorie comportementale permet de simuler un système d'événements avec des objets. Son comportement est similaire à des callbacks ou des webhooks.

Ce design pattern est composé de deux parties :

-une première qui définit le comportement des objets qui « observent ».

-une deuxième qui définit le comportement des objets qui sont « observables ».

De ces deux parties découlent une classe et une interface ; l'une se nomme IObserver et l'autre Observable

Exo 1 :

Votre rôle est de compléter l'implémentation de Observable.cpp.

Stratégie :

Ce design pattern de la catégorie comportementale permet d'assigner des comportements de manière dynamique.

L'architecture de ce design pattern se repose sur une Interface IStrat qui va définir le contenu des stratégies à appliquer avec la fonction process.

Exo 2 :

Dans cet exercice vous devez réimplémenter l'interface IStrat pour qu'elle soit compatible avec les stratégies qui en héritent.

Visiteur :

Ce design pattern de la catégorie comportementale permet d'étoffer le comportement d'une classe sans ma modifier et dans une moindre mesure, d'utiliser le polymorphisme pour appliquer certains comportements à certaines classes en particulier.

Ce modèle est découpé en plusieurs parties :

-les classes « fermées »

-la classe abstraite Visitor qui définit les visiteurs qui en hériteront.

-les visiteurs qui définissent un comportement spécial à appliquer sur chaque type d'objet.

Exo 3 :

Pour ce dernier exercice, il vous est demandé de compléter deux choses :

-dans visitors.hpp, vous devez complétez les classes des visiteurs.

-dans Usefull.hpp et Useless.hpp, vous devez compléter la méthode accept qui va recevoir des visiteurs pour s'envoyer elle-même à celui-ci.