Membres : Schaffo Raphaël, Ryter Nils, Vulliemin Kevin

Equipe : Numéro 6

Destinataire : Monsieur Gobron Stéphane

# Introduction

## Problématique

Dans le cadre du Design Pattern, nous devons développer 3 programmes distincts pour le laboratoire 3 afin d’étudier les patterns « Composite » (vu au laboratoire 2), « Decorator » et « State ».

### Tâche 1

Le premier programme doit permettre à l’utilisateur de **composer** des paniers de fruits. Par là on entend des actions tels que :

* Créer des fruits ;
* Ajouter des fruits dans un panier ;
* Ajouter des paniers dans des paniers ;
* Selon développement, éventuellement suppression des fruits et paniers

L’affichage des paniers produirait les lignes suivant (avec 1 panier composé d’une banane, d’une poire et de 2 fraises) :

1. Panier[Banane, Poire, Fraise, Fraise] – Sans pépin

### Tâche 2

Le deuxième programme peut être implémenté sur la structure du premier. Il s’agit de produire un affichage de nos paniers de façon à ce que ceux-ci renferme nos fruits « graphiquement » et les **décorent**.

Exemple basé sur le panier de la rubrique précédente:

1. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. Banane, Poire, Fraise, Fraise
3. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Tâche 3

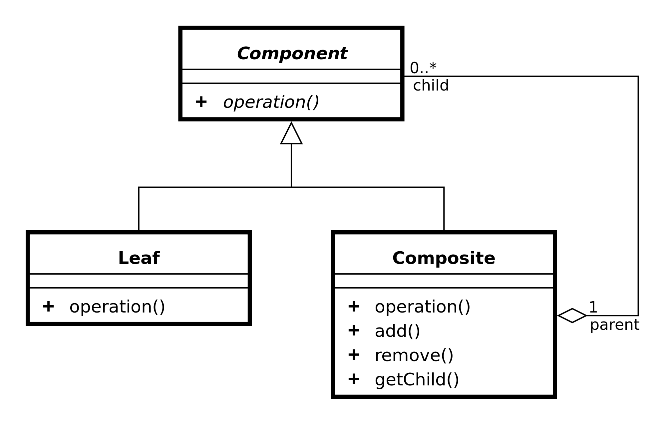
Cette dernière tâche consiste essentiellement à traiter une commande d’un **état** à un autre. Le déroulement séquentiel consiste en la suite donné :

Pour passer de l’état « En attente » à « Payée » on devra simuler le payement par un clic de l’utilisateur. Le statut « Envoyé » sera automatiquement appelé une fois la condition « Payée » atteinte.

# Conception

## Composite

La structure composite ayant déjà été abordée dans le laboratoire 2, on n’abordera pas sa structure dans ce rapport. On comprendra simplement que le Fruit et l’objet feuille et qu’il sera compris dans un panier physique qui est le composite. Comme il n’existe pas d’objet abstrait confondant des fruits et des paniers de fruits, on l’appellera tout simplement « Component ».



## Decorator

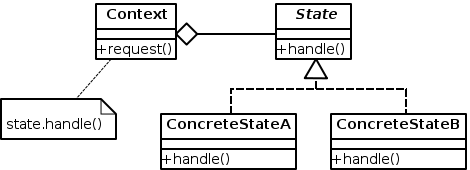
On voit tout de suite la ressemblance de structure entre le pattern « Decorator » et « Composite » :



On peut donc bien utiliser la structure créée dans la tâche 1. L’objet « Decorator » correspondant n’est autre que le panier que l’on va tout simplement étendre (créer des enfants de la classe « Decorator ») afin de faire des paniers de « # », des paniers de « \* » ou de « = ». Ce seront les « ConcreteDecorator ».

## State

Le pattern « State » est l’abstraction des comportements possibles d’un objet sous forme d’une classe selon ces états.



On va tout simplement créer un objet pour chaque état possible héritant de la classe abstraite « State ».

# Implémentation

## Séparation

Il a été décidé qu’on séparerait les 3 tâches dans 3 projets différents afin de distinguer les patterns utilisés pour chacun et garantit la compréhension de l’évolution de la tâche 1 vers la 2.

## Tâche 1 – Composite

La fonction qui se transmet aux enfants sera simplement la fonction d’affichage.

## Tâche 2 – Decorator

On créera simplement nos classes enfants comme « DecoratorEtoile » et les autres demandés en héritant de la classe « Decorator » et de son opération en prenant le soin de la redéfinir.

## Tâche 3 – State

On va simplement attribuer des états lors du traitement et les attribuer à l’objet. Dans notre cas on supprimera l’objet instancié et on en recréera un objet état correspondant à attribuer.

# Conclusion

Tous les programmes sont fonctionnels. Les 2 premières tâches produisent à l’écran les résultats attendus dans l’introduction. Le dernier programme affiche un suivi des opérations pour plus de clarté dans les changements d’états. L’utilisation de chaque pattern était justifiée dans son exercice.