RAKOTONARIVO MIANDRISOA SANDRATRA ESIIA 5 N°12

ANDRIANJAFIMAHEFARINJO Manoa Fitiavana N°8

HARIVELO Tantely Lalaina N°6

Explication Décorateur

1. Introduction

Le Décorateur est un patron de conception structurel qui permet d’ajouter dynamiquement de nouveaux comportements à des objets en les plaçant à l’intérieur d’objets spéciaux appelés emballeurs (wrappers).

Le décorateur est également appelé « emballeur » ou « empaqueteur ». Ces surnoms révèlent l’idée générale derrière le concept. Un emballeur est un objet qui peut être lié par un objet cible. L’emballeur possède le même ensemble de méthodes que la cible et lui délègue toutes les demandes qu’il reçoit. Il peut exécuter un traitement et modifier le résultat avant ou après avoir envoyé sa demande à la cible.

Un emballeur implémente la même interface que l’objet emballé. Du point de vue du client, ces objets sont identiques. L’attribut de la référence de l’emballeur doit pouvoir accueillir n’importe quel objet qui implémente cette interface. Vous pouvez ainsi utiliser plusieurs emballeurs sur un seul objet et lui attribuer les comportements de plusieurs emballeurs en même temps.

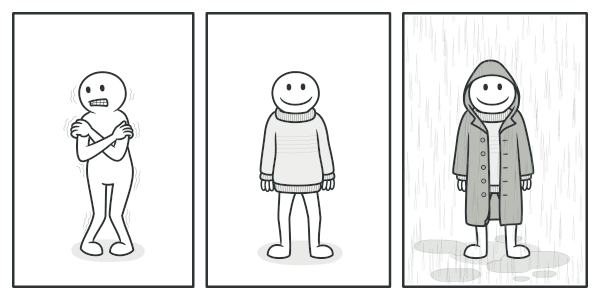
Un décorateur peut décorer un composant.

* Problème :

Surdimensionné le problème / les attributs.

Explosion de classe.

Fonctionnalité non approprié pour certains sous classe



Les effets se cumulent si vous portez plusieurs couches de vêtements.

Porter des vêtements est un bon exemple d’utilisation. Si vous avez froid, vous vous enroulez dans un pull. Si vous avez encore froid, vous pouvez porter un blouson par-dessus. S’il pleut, vous enfilez un imperméable. Tous ces vêtements « étendent » votre comportement de base mais ne font pas partie de vous, et vous pouvez facilement enlever un vêtement lorsque vous n’en avez plus besoin.

## Possibilités d’application

* Utilisez le décorateur si vous avez besoin d’ajouter des comportements supplémentaires au moment de l’exécution sans avoir à altérer le code source de ces objets.
* Le décorateur vous permet de structurer votre logique métier en couches, de créer un décorateur pour chacune de ces couches et de décorer les objets avec différentes combinaisons au moment de l’exécution. Le code client peut traiter les objets uniformément puisqu’ils implémentent la même interface.
* Utilisez ce patron si l’héritage est impossible ou peu logique pour étendre le comportement d’un objet.
* De nombreux langages de programmation permettent l’utilisation du mot clé final pour interdire l’héritage d’une classe. Le seul moyen d’étendre le comportement d’une telle classe est de l’emballer en utilisant un décorateur.

1. Mise en œuvre
2. Assurez-vous que votre domaine peut être représenté sous la forme d’un composant principal recouvert par plusieurs couches facultatives.
3. Déterminez les méthodes communes entre le composant principal et les couches facultatives. Créez l’interface du composant et déclarez-y ces méthodes.
4. Créez une classe concrète pour le composant et définissez son comportement de base.
5. Créez une classe de base décorateur. Elle doit inclure un attribut qui va permettre de stocker la référence à un objet emballé. Cet attribut doit être déclaré avec le type de l’interface du composant, afin de le relier aux composants concrets et aux décorateurs. Le décorateur de base doit déléguer tout le travail à l’objet emballé.
6. Assurez-vous que les classes implémentent l’interface du composant.
7. Créez des décorateurs concrets en les implémentant à partir du décorateur de base. Un décorateur concret doit exécuter son comportement avant ou après l’appel à la méthode de son parent (qui délègue toujours la tâche à l’objet emballé).
8. Le code client doit être responsable de la création des décorateurs et de leur agencement en fonction des besoins du client.
9. Avantages et inconvénients
10. Avantages

* Vous pouvez étendre le comportement d’un objet sans avoir recours à la création d’une nouvelle sous-classe.
* Vous pouvez ajouter ou retirer dynamiquement des responsabilités à un objet au moment de l’exécution.
* Vous pouvez combiner plusieurs comportements en emballant un objet dans plusieurs décorateurs.
* Principe de responsabilité unique. Vous pouvez découper une classe monolithique qui implémente plusieurs comportements différents en plusieurs petits morceaux.

1. Inconvénients

* Retirer un emballeur spécifique de la pile n’est pas chose aisée.
* Il n’est pas non plus aisé de mettre en place un décorateur dont le comportement ne varie pas en fonction de sa position dans la pile.
* Le code de configuration initial des couches peut avoir l’air assez moche.