Projet Java IV

2023

Une image contenant texte, clipart

Description générée automatiquement

Réalisé par Popadiuc Claudiu

Supervisé par Monsieur Riggio, Jonathan

Table des matières

[Introduction 3](#_Toc134661574)

[Présentation de l’interface graphique 4](#_Toc134661575)

[Analyse et applications des Design Patterns 5](#_Toc134661576)

[Limitations 6](#_Toc134661577)

[Conclusion 7](#_Toc134661578)

# **Introduction**

# **Présentation de l’interface graphique**

# **Analyse et applications des Design Patterns**

* **Factory** : Permet de centraliser la création de mes objets. J’utilise ce Design Pattern pour créer mes composants et mes produis.
* **Singleton** : Permet de créer une seule instance de la classe via la methode getInstance (). Je l’utilise dans la factory car.
* **MVC** : J’ai utilisé la classe **« HELBElectroView »** qui représente la vue, la classe « HELBElectroController » qui sert de Controller et les autres classes servent pour le modèle.
* **Créateur** : Qui détient la responsabilité de créer la vue de mon interface graphique avec tout ce qui se tient de la Grid, Vbox, Boutton, etc... La classe **« HELBElectroView »** a la responsabilité de création de tout cela car c’est cette classe qui s’occupe de l’affichage.
* Créateur : Qui détient la responsabilité de créer les boutons ? La classe « InterfaceGraphic » a la responsabilité de création des boutons car c’est cette classe qui s’occupe de l’affichage.
* Expert en information : principe d’attribuer la responsabilité aux objets. Le fait que les emplacements remplissent le parking. Le parking contient toutes les informations des emplacements donc c’est logique qu’elles soient créées dans celui-ci (ControllerParking).
* Principe de faible couplage : si on effectue une modification, cela ne devrait pas impacter d’autres classes. Il faut réduire au minimum le nombre de modification à effectuer.
* MVC : j’ai utilisé la classe « IntrefaceGraphic » qui représente la vue, la classe « ControllerParking » qui sert de Controller et les autres classes servent pour le modèle.
* Contrôleur : la classe « ControllerParking ». Elle représente le système dans sa globalité. Par exemple, le fait que lorsqu’on ajoute un véhicule et que l’emplacement change de couleur, cela crée des changements dans les données.
* Singleton : permet de créer une seule instance de classe.
* Fabrique : permet de centraliser la création de mes objets. J’utilise ce pattern pour créer mes véhicules. S’il y a d’autres objets à créer, c’est dans cette classe que ça se passe.
* Stratégie : le comportement des prix varie selon les jours. Ils évoluent donc dynamiquement à l’exécution.

# **Limitations**

# **Conclusion**