Projet Java IV

2023

Une image contenant logo

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**

Dans le cadre du cours de JAVA Q IV, nous avons été chargés de développer un logiciel de gestion d'entrepôt. Ce logiciel est conçu pour gérer plusieurs entrepôts automatisés qui s'occupent à la fois de la fabrication et du stockage de produits électroniques, le tout supervisé par un employé qualifié.

Dans ce rapport, je vais commencer par présenter l'interface graphique de manière visuelle et explicative afin de faciliter la compréhension du fonctionnement de l'application.

Ensuite, j'introduirai l'analyse et les design patterns utilisés dans l'application, en expliquant la structure de ma mise en œuvre à l'aide d'outils d'analyse. Je détaillerai également les raisons pour lesquelles j'ai choisi certains design patterns spécifiques pour mon application.

Ensuite, j'aborderai les limitations de mon application et me poserai des questions telles que dans quels cas d'utilisation l'application pourrait-elle ne pas fonctionner comme prévu ? Y a-t-il des aspects techniques qui n'ont pas été pris en compte ? Si j'avais disposé de plus de temps, qu'aurais-je pu améliorer ?

Enfin, je conclurai ce rapport en résumant les principales conclusions et en soulignant les points clés du développement du logiciel de gestion d'entrepôt.

# **Présentation de l’interface graphique**

Une image contenant capture d’écran, Rectangle, carré, Caractère coloré

Description générée automatiquement

Permettez-moi de vous présenter l'interface graphique du logiciel de gestion d'entrepôt. Comme vous pouvez le constater, la production de produits et de composants est déjà en cours depuis un certain temps. Dans la partie gauche de l'interface, vous pouvez voir la zone de stockage des produits. Nous avons déjà plusieurs types de produits, tels que le P7, qui est un drone, ainsi que d'autres types de produits existants.

Dans la partie droite de l'interface, vous trouverez la zone de stockage des composants. Nous avons également un composant disponible appelé le "C-Type-2". Grâce à ce composant, il est possible de fabriquer différents produits, d'où la présence des produits déjà fabriqués dans la partie gauche de l'interface.

Cette visualisation de l'interface graphique nous permet d'avoir une vue d'ensemble claire de l'état actuel de l'entrepôt. Elle facilite la compréhension du flux de production et du stockage des produits électroniques. En examinant les différentes zones de stockage et les produits déjà présents.

Une image contenant texte, capture d’écran, Système d’exploitation, Police

Description générée automatiquement

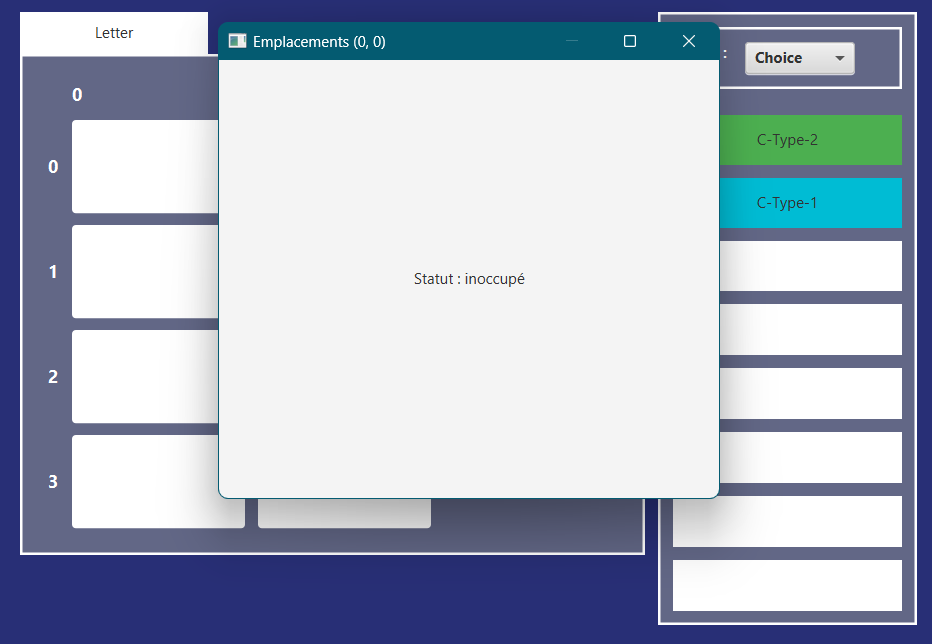
Si la partie de gauche, la zone de sotckage des produits alors on recoit une alerte qui nous le previent, il suffit ensuite d’appuyer sur « ok » pour vider tout l’entrepot en une seul fois.

Une image contenant capture d’écran, texte, Rectangle, diagramme

Description générée automatiquement

Une image contenant capture d’écran, Rectangle, carré, fenêtre

Description générée automatiquement



Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Description générée automatiquement

Une image contenant capture d’écran, Rectangle, carré, Caractère coloré

Description générée automatiquement

# **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**