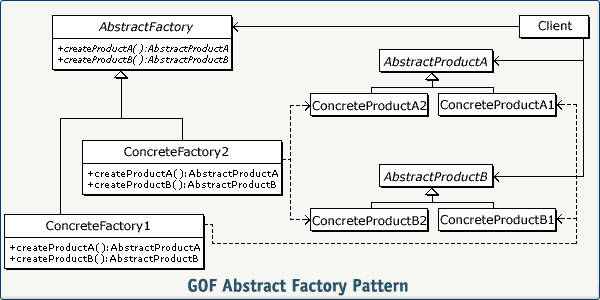
Fabriques Abstraites

(AbstractFactory)



* Ce modèle fournit une interface pour créer des familles d’objets liées ou dépendants sans spécifier leur classes concrètes
* Il délègue la création des objets produits à sa sous-classe **ConcreteFactory**
* Isolation des classes concrètes : facilite le contrôle des classes des objets créés par une application
* Encapsule les actions de prise en charge/création des objets produits (isole les clients des classes d’implémentation)
* Clients manipulent les instances à travers leurs interfaces abstraites
* Les noms des classes produits sont isolés dans l’implémentation de la **ConcreteFactory** (n’apparaissent pas dans le code client)
* Substitution de familles de produits facilités
* Une **ConcreteFactory** n’apparaît qu’une seule fois dans une application (là où elle est instanciée)

→ Il est alors facile de la modifier

* Une **AbstractFactory** engendre une famille complète de produits

→ Si on la modifie : c’est la totalité de la famille de produits qui est remplacée

* Favorise le maintien de la cohérence entre les produits
* Quand les objets produits d’une famille sont destinés à travailler ensemble il est important qu’une application n’utilise les objets d’une seule famille à la fois

→ La classe **AbstractFactory** facilite le renforcement de cette condition

* Contraintes : supporter de nouveaux types de produits est difficile
* L’interface **AbstractFactory** détermine l’ensemble des produits qui peuvent être créés
* Supporter de nouveaux types de produits requiert l’extension de l’interface de la fabrique

→ Implique la modification de la classe **AbstractFactory** et de toutes ses sous-classes

* Implémentation
* Utiliser les fabriques en tant que singletons (une seule instance/un seul point d’accès vers elle)

→ Une application ne nécessite qu’une seule instance d’une **ConcreteFactory** par famille de produits

* **AbstractFactory** déclare juste une interface pour la création des produits

→ C’est aux sous-classes **ConcreteProduct** de les créer (définir une méthode de fabrication pour chaque produit)

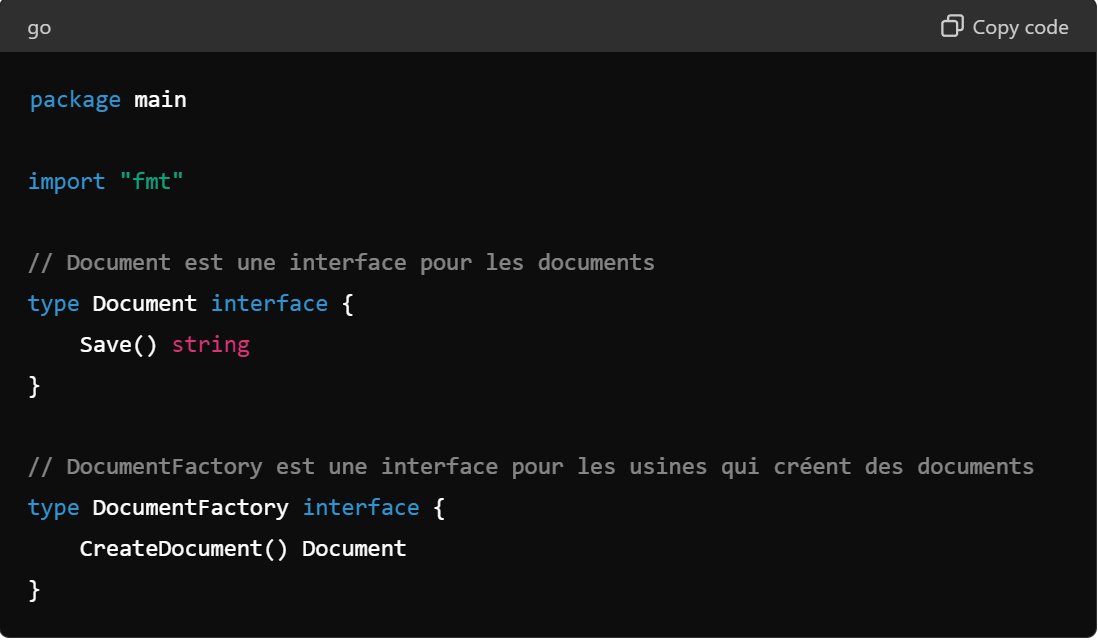
→ **ConcreteFactory** spécifiera ses produits par override (surcharge) pour chacun d’eux de la méthode de fabrication

* Utilisation d’**AbstractFactory**
* Un système doit être indépendant de la manière dont ses produits ont été créés/combinés/représentés
* Un système doit être constitué à partir d’une famille de produits parmi plusieurs
* Renforcer le caractère de communauté d’une famille d’objets produits conçus pour être utilisés ensemble
* Fabriquer une bibliothèque de classes de produits en ne révélant que l’interface et non l’implémentation
* Constituants d’**AbstractFactory**
* **AbstractFactory** : déclare une interface contenant les opérations de création d’objets produits abstraits
* **ConcreteFactory** : implémente les opérations de création d’objets produits concrets
* **AbstractProduct** : Déclare une interface pour un objet de type produit
* **ConcreteProduct**: Définit un objet produit créé par la **ConcreteFactory** correspondante
* **Le client**: n’utilisera que les interfaces **AbstractFactory** et **AbstractProduct**
* Collaboration d’**AbstractFactory** (délègue la création des objets produits à sa sous-classe **ConcreteFactory**)
* Une seule **ConcreteFactory** est crée à l’exécution (crée des produits dotés d’une implémentation spécifique)
* Pour créer des objets produits différents les clients doivent utiliser une **ConcreteFactory** différente

Exemples

1. Générer différents types de documents sans modifier le code client (pdf/world)

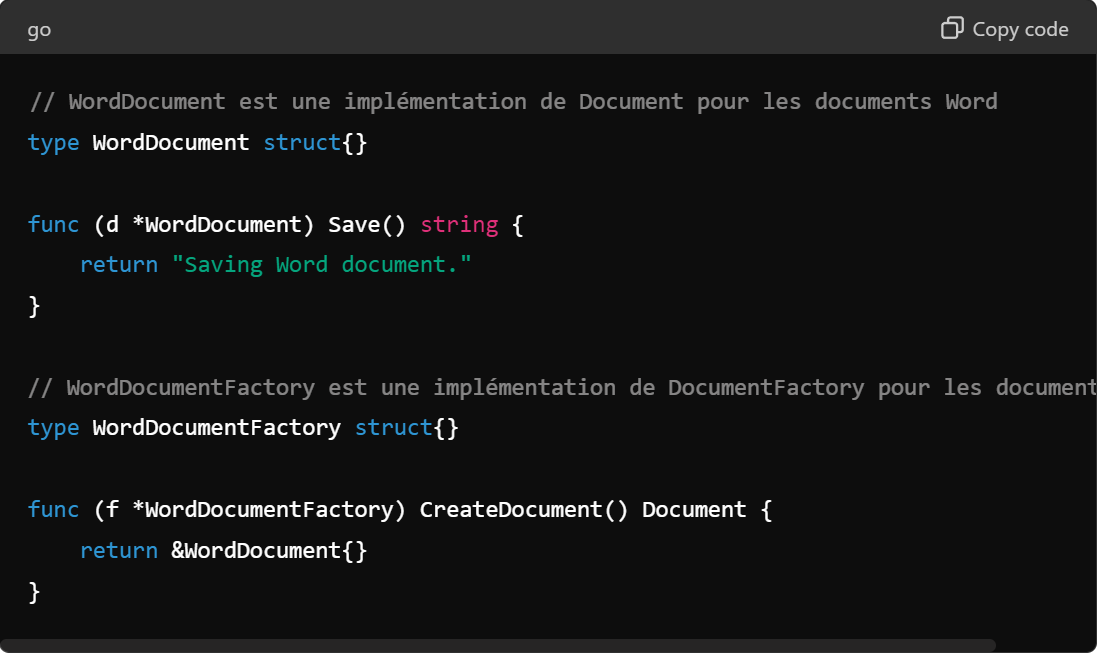
Interfaces et structures abstraites



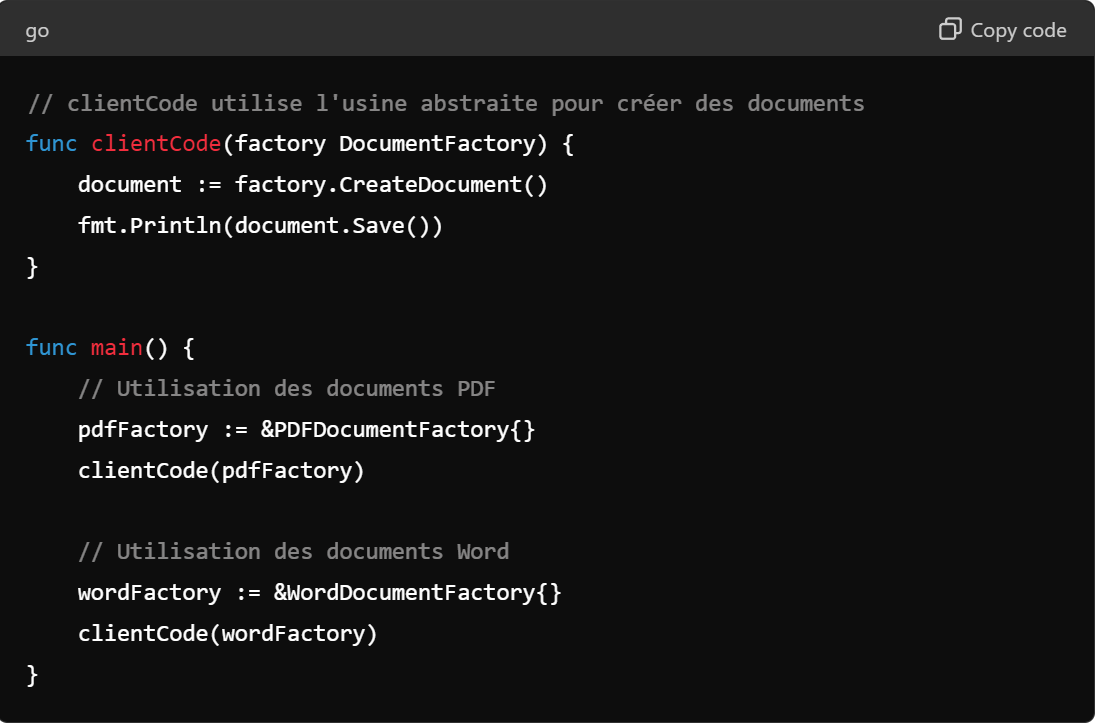
Classes concrètes pour pdf



Classes concrètes pour World

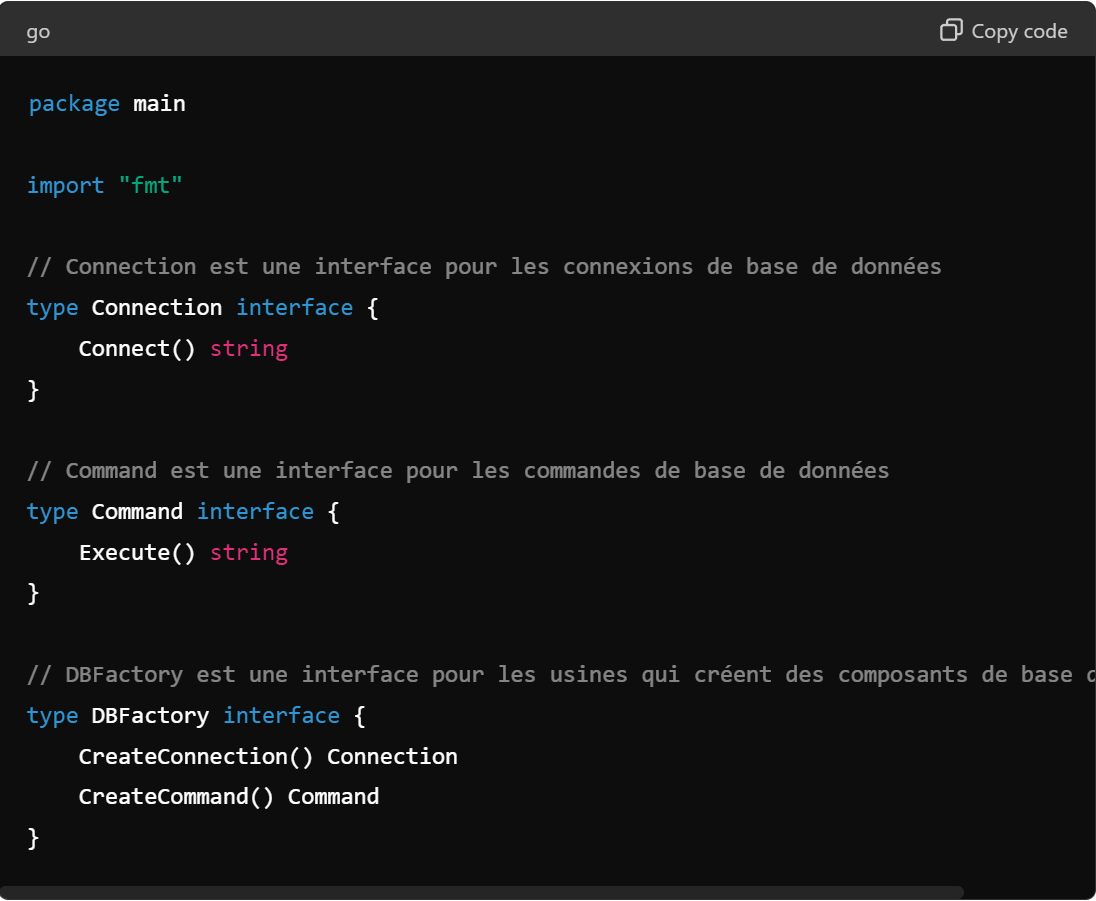


Utilisation du client



1. Système de base de données

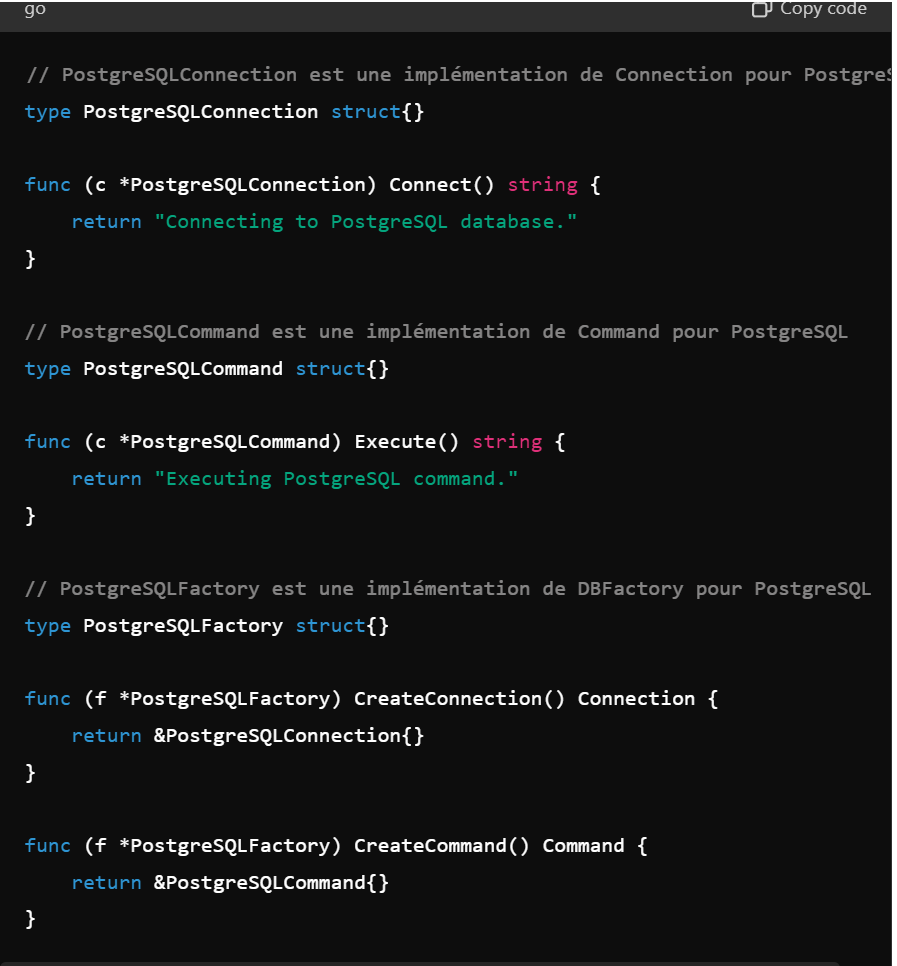
Interfaces et structures Abstraites



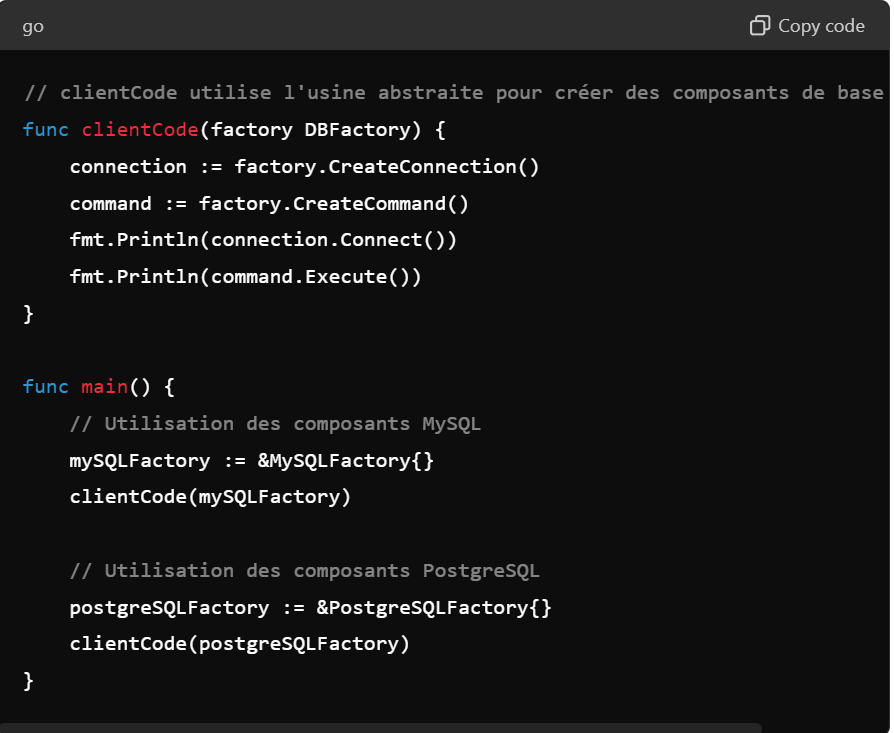
Classe concrètes pour MySQL



Classes concrètes pour PostgresSQL

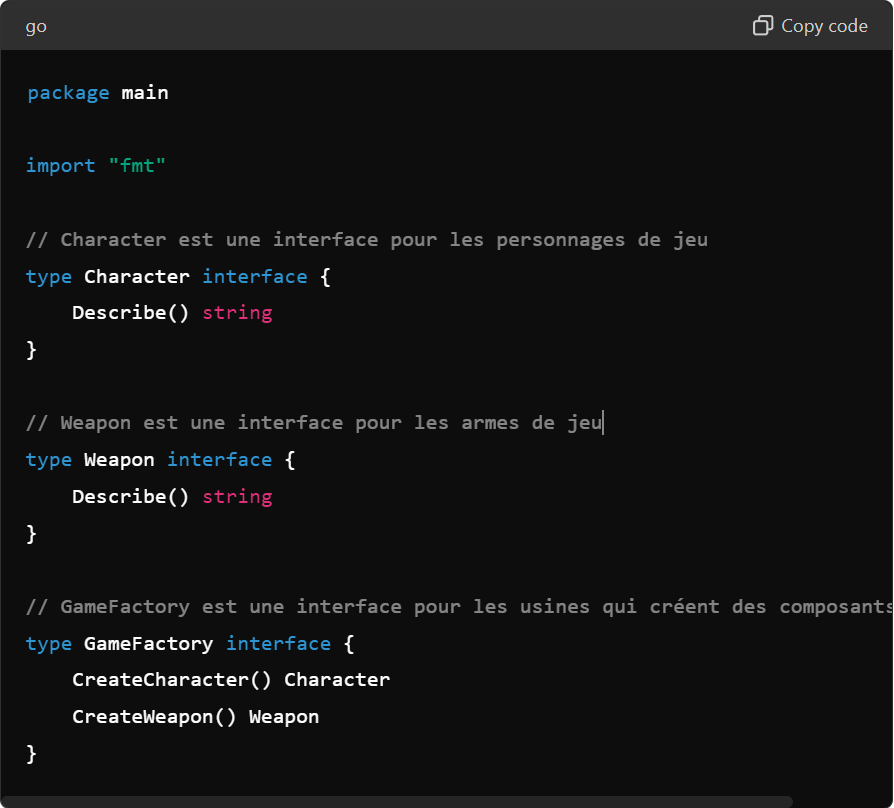


Utilisation du client

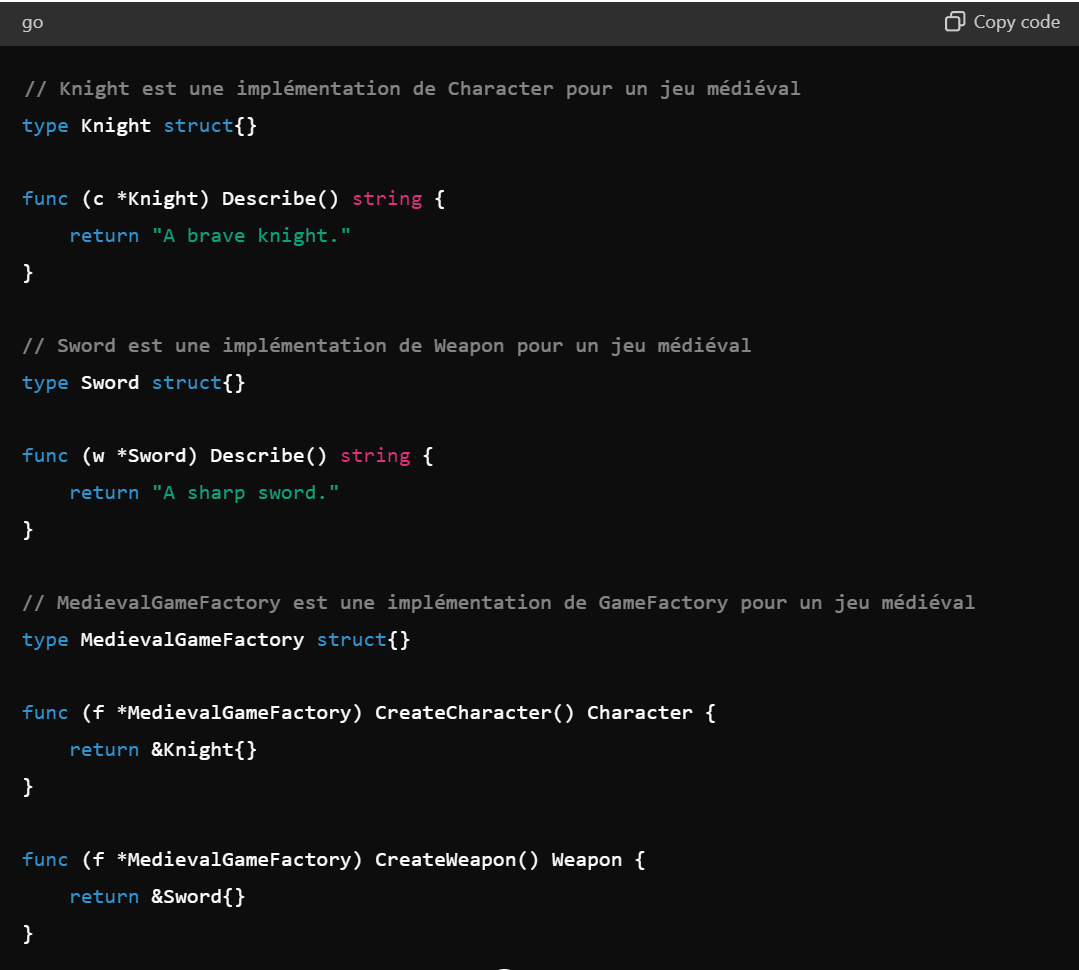


1. Jeux vidéo : créer objets de jeu (personnages/armes) pour différents types de jeux (médiéval/futuriste)

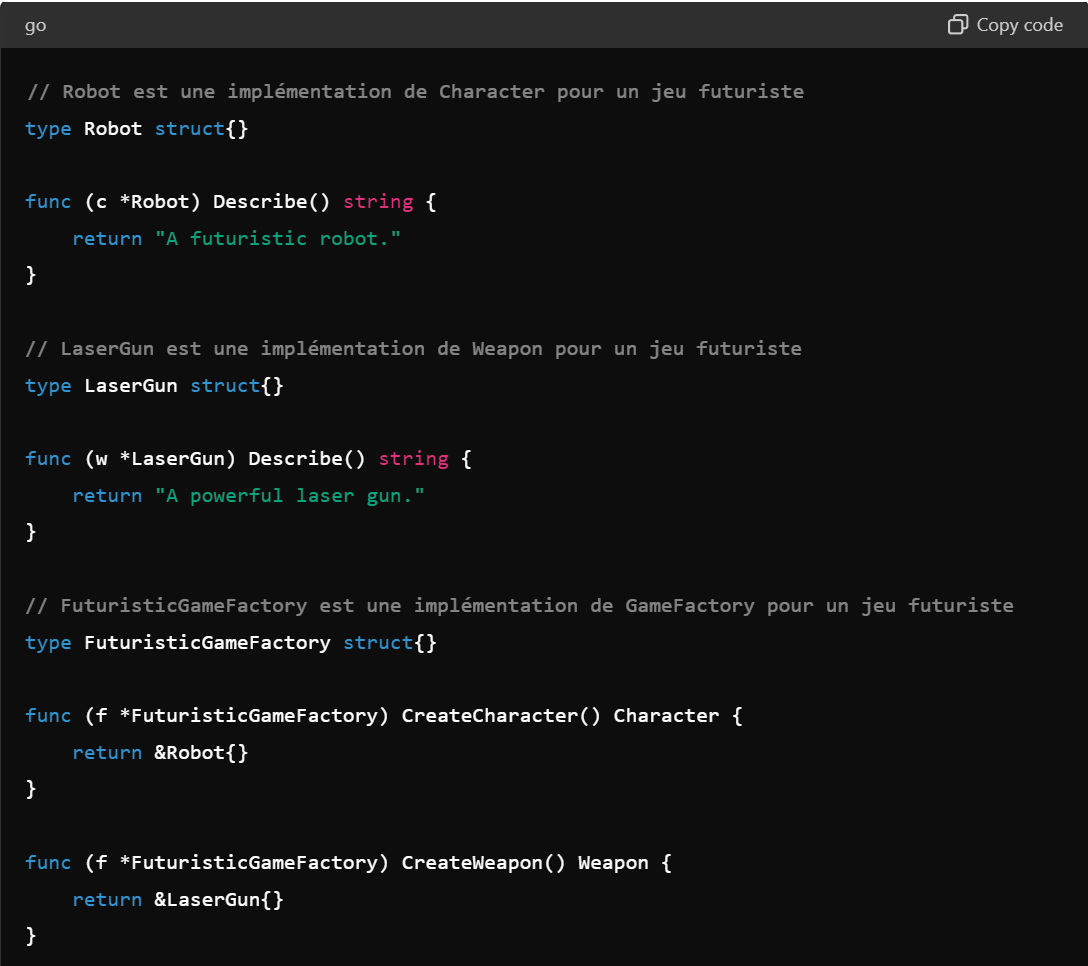
Interfaces et structures Abstraites



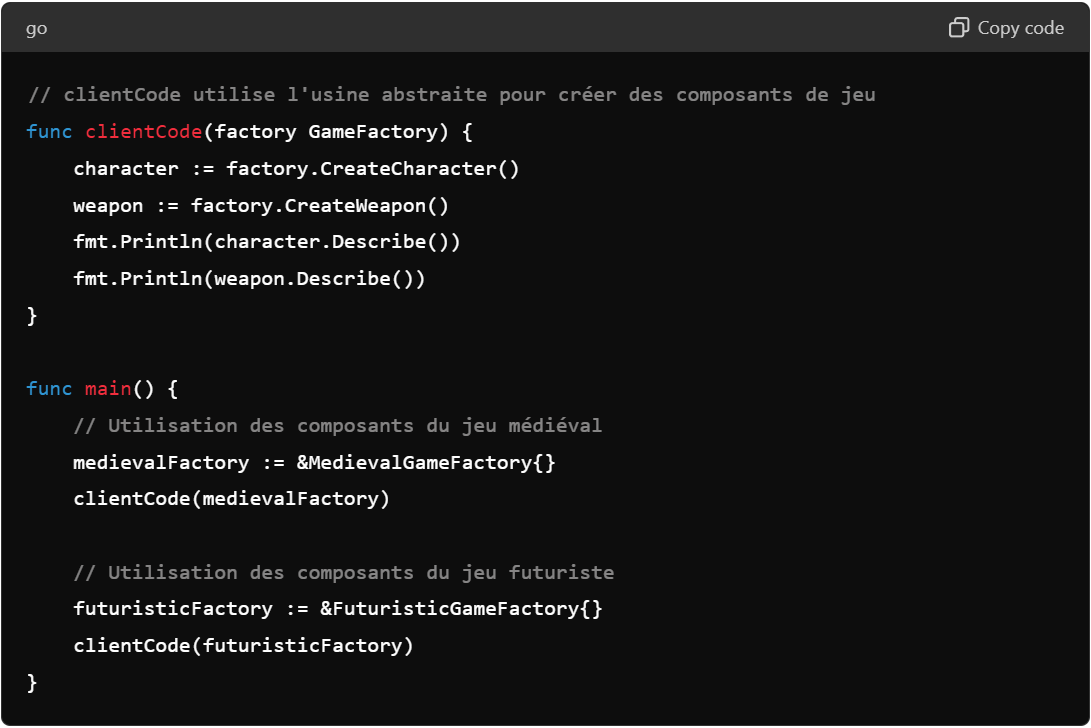
Classes concrètes pour un jeu médiéval



Classes concrètes pour un jeu futuriste



Utilisation client



AbstractFactory est utile pour découpler le code client des implémentations spécifiques de différents objets

* Permet de créer des systèmes modulaires/extensibles où il est facile de changer/ajouter de nouvelles familles d’objet sans modifier le code client