Début du projet concessionnaire automobile

Pour ces exercices, nous allons réutiliser le code de l'exercice précédent, pour l'améliorer.

```go
package main

import (
    "fmt"
    "time"
)

// Structure Car
type Car struct {
    brand  string
    year   int
    color  string
    engine int
}

// Fonction receiver Equals pour la structure Car
func (c Car) Equals(other Car) bool {
    return c.brand == other.brand && c.year == other.year
}

// Fonction receiver Age pour la structure Car
func (c Car) Age() int {
    return time.Now().Year() - c.year
}

// Fonction receiver UpdateYear pour la structure Car
func (c *Car) UpdateYear(newYear int) *Car {
    c.year = newYear
    return c
}

// Fonction receiver UpdateColor pour la structure Car
func (c *Car) SetColor(newColor string) *Car {
    c.color = newColor
    return c
}

// Fonction receiver UpgradeEngine pour la structure Car
func (c *Car) UpgradeEngine() *Car {
    c.engine++
    return c
}

func main() {
    myCar := Car{brand: "Toyota", year: 2010}
    fmt.Printf("La voiture a %d ans.", myCar.Age())
    myCar.UpdateYear(2009)
    fmt.Printf("La voiture a %d ans.", myCar.Age())

    // Comparaison de deux voitures
    myCar2 := Car{brand: "Toyota", year: 2010}
```

```
    fmt.Println(myCar.Equals(myCar2))
    myCar2.UpdateYear(2009)
    fmt.Println(myCar.Equals(myCar2))

    // Mise à jour chaînée
    myCar.UpdateYear(2021).SetColor("red").UpgradeEngine()
    fmt.Println(myCar)
}
```

Exercice 1 : Split dans des Packages Différents

> ***Objectif :*** *Factorisez le code pour séparer les structures et les fonctions dans des packages distincts.*

Instructions :

- Créer un package `car` et y déplacer la structure `Car` et toutes ses méthodes.
- Enlevez les retours de type des méthodes `UpdateYear`, `SetColor` et `UpgradeEngine`.
- Ajouter une fonction `New(brand string, year int, color string, engine int) *Car` qui crée une nouvelle instance de `Car` et l'initialise avec les valeurs passées en paramètre.
- Créer un package `main` où vous appellerez les fonctions du package `car`.
- Assurer que le programme fonctionne comme avant après la factorisation.

Résultats attendus :

Un programme modulaire où la logique liée à Car est isolée dans son propre package.

Exercice 2 : Gestion d'Erreur

> ***Objectif :*** *Ajouter de la gestion d'erreur dans les méthodes de Car.*

Instructions :

- Modifier la méthode `UpdateYear` pour qu'elle renvoie une erreur si l'année est dans le futur.
- Gérer l'erreur dans la fonction `main` et afficher un message approprié.
- Ajouter des validations similaires dans d'autres méthodes si nécessaire (par exemple, `SetColor` ne devrait pas accepter une chaîne vide).

Résultats attendus :

Un programme qui gère correctement les cas d'erreur.

Exercice 3 : Ajouter du Logging

> ***Objectif :*** *Intégrer un système de logging pour suivre les opérations effectuées sur les instances de Car.*

Instructions :

- Ajouter du **logging** dans chaque méthode de `Car` pour indiquer quelle opération a été effectuée.
- Les **logs** doivent inclure des informations pertinentes comme l'état avant et après l'opération.

Résultats attendus :

Un programme qui affiche des logs détaillés des opérations effectuées.

Exercice 4 : Projet Concessionnaire Automobile

> ***Objectif :*** *Transformer progressivement le code pour qu'il simule un système de gestion pour un concessionnaire automobile.*

Exercice 4a : Ajout de Voitures au Concessionnaire

> ***Objectif** : Créer une fonction pour ajouter des voitures à un inventaire.*

Instructions :

- Créer une structure `Dealership` qui contiendra un slice de `Car`.
- Ajouter une méthode `New() *Dealership` à `Dealership` qui crée une nouvelle instance de `Dealership` et l'initialise avec un slice vide.
- Ajouter une méthode `AddCar(car car.Car)` à `Dealership` pour ajouter une voiture à l'inventaire.
- Dans la fonction `main`, créer une instance de `Dealership` et ajouter quelques voitures.

Signature de la fonction :

```
func (d *Dealership) AddCar(car Car)
```

Conseils :

`Dealership` peut être défini dans un package différent de `Car`.

Résultats attendus :

Un programme capable d'ajouter des voitures à un inventaire de concessionnaire.

Exercice 4b : Suppression de Voitures de l'Inventaire

> ***Objectif :** Créer une fonction pour supprimer des voitures de l'inventaire.*

Instructions :

- Ajouter une méthode `RemoveCar(index int) error` à `Dealership` pour supprimer une voiture à un index spécifié.
- Gérer les erreurs en cas d'index invalide.
- Tester la suppression dans la fonction `main`.

Signature de la fonction :

```
func (d *Dealership) RemoveCar(index int) error
```

Résultats attendus :

Un programme capable de gérer la suppression de voitures de l'inventaire.

Exercice 4c : Recherche de Voitures par Marque et Année

> ***Objectif :** Implémenter une fonctionnalité permettant de rechercher des voitures dans l'inventaire en fonction de leur marque et année de fabrication.*

Instructions :

- Dans la structure `Dealership`, ajoutez une méthode `FindCarsByCriteria(criteria map[string]string) []car.Car` pour rechercher des voitures dans l'inventaire.
- La méthode doit accepter un `map` de critères de recherche.
  - Chaque clé du `map` correspond à un critère de recherche (brand, year, color, engine).
- La méthode doit parcourir l'inventaire et retourner un slice de `Car` qui correspond aux critères de recherche.
- Les critères de recherche peuvent être combinés (par exemple, rechercher des voitures de marque Toyota et de couleur rouge).

- Pour accéder aux propriétés de la voiture, vous pouvez utiliser les méthodes `GetBrand()`, `GetYear()`, `GetColor()` et `GetEngine()`, qu'il faudra ajouter à la structure `Car`.
- Assurez-vous de gérer le cas où aucune voiture ne correspond aux critères (retourner un slice vide).
- Testez cette fonctionnalité dans la fonction `main` en recherchant des voitures avec différents critères.

Signature de la fonction :

```
func (d *Dealership) FindCarsByCriteria(criteria map[string]string) []car.Car
```

Exemple d'utilisation :

```
criteria := map[string]string{
    "brand": "Toyota",
    "year": "2010",
}

cars := dealership.FindCarsByCriteria(criteria)
```

Résultats attendus :

Un programme capable de rechercher et d'afficher des voitures de l'inventaire basé sur des critères de recherche.

Exercice 4d : Affichage de l'Inventaire

> ***Objectif :*** *Créer une fonction pour afficher l'intégralité de l'inventaire du concessionnaire.*

Instructions :

- Ajouter une méthode `DisplayInventory()` à `Dealership` qui affiche toutes les voitures de l'inventaire.
- Assurer que l'affichage est clair et bien formaté.
- Tester l'affichage dans la fonction `main`.

Signature de la fonction :

```
func (d *Dealership) DisplayInventory()
```

Résultats attendus :

Un programme capable d'afficher l'inventaire de voitures de façon claire et organisée.