Les bases du langage Go
Partie 4 : interfaces

Initiation au développement

BUT informatique, première année

Ce TP a pour but de vous présenter les bases du langage Go. C'est ce langage qui sera utilisé pour réaliser tous les exercices de ce cours d'initiation au développement. Il n'y a pas de prérequis particuliers en dehors du contenu des TP précédents (les bases du langage Go partie 1, partie 2 et partie 3).

Dans tout ce TP un texte avec cet encadrement est un texte contenant une information importante, il faut donc le lire avec attention et veiller à en tenir compte.

Un texte avec cet encadrement est une remarque.

Exercice 4.0.0 : exemple d'exercice
Un texte avec cet encadrement est un travail que vous avez à faire.

Même si la plupart des exercices de ce TP peuvent vous sembler très faciles, faites les tous sérieusement : cela vous permettra de retenir la syntaxe du langage Go et d'être ainsi plus à l'aise lors des prochains TPs.

Si vous souhaitez avoir un autre point de vue sur l'apprentissage du langage Go, vous pouvez consulter le tutoriel officiel *A Tour of Go*[1] ou le site *Go by Example*[2] en faisant toute fois attention au fait que ces sites sont plutôt faits pour les personnes ayant déjà une bonne expérience de la programmation.

1. `https://tour.golang.org/`
2. `https://gobyexample.com/`

1 Mise en pratique des structures et méthodes

On suppose qu'on souhaite gérer les notes de différentes promotions d'étudiants, calculer les moyennes des étudiants et les classer. Pour cela on va définir un type pour nos étudiants et un type pour une promo (qui contiendra plusieurs étudiants). On définira ensuite un certain nombre de méthodes sur les étudiants et sur les promos.

Exercice 4.1.0
Définir un type Etudiant comprenant un nom, un prénom et une liste de notes.

Exercice 4.1.1
Définir une méthode Moyenne qui s'applique à un étudiant et calcule sa moyenne.

Exercice 4.1.2
Définir une méthode AjouteNote qui s'applique à un étudiant et ajoute une note dans sa liste de notes.

Exercice 4.1.3
Définir un type Promotion comprenant un nom de promo et une liste d'étudiants.

Exercice 4.1.4
Définir une méthode Classer qui s'applique à une promotion et classe les étudiants en fonction de leur moyenne.

2 Pour aller plus loin : interfaces

Les interfaces permettent de définir des fonctions génériques, qui vont s'appliquer à plusieurs types d'éléments, à condition que ces types partagent un ensemble de méthodes (qui seront utilisées pour les manipuler).

Nous avons déjà utilisé des interfaces sans le savoir quand nous avons créé des bufio.Reader. Ceux-ci peuvent être créés à partir de n'importe quelle variable d'un type qui met en œuvre l'interface io.Reader, c'est-à-dire qui a une méthode Read.

Une interface est en fait simplement une liste de signatures de méthodes avec un nom. On dit qu'un type qui possède toutes ces méthodes implante l'interface. Dès qu'une fonction prend en paramètre le type de l'interface on peut utiliser à la place tout type qui implante l'interface. Le programme 1 montre un exemple de définition et d'implantation d'interface.

Ce programme définit une interface Forme qui décrit une forme générale avec un périmètre. Sur cette interface, une fonction est définie, permettant de comparer les périmètres de deux formes.

Ensuite, deux implantation de l'interface forme sont proposées : par le type Carre et par le type Cercle.

Dans le main on voit ensuite qu'on peut comparer les périmètres de carres et de cercles, en utilisant les fonctions qui ont été définies à partir de l'interface Forme.

Exercice 4.2.0
Récupérez le fichier interface.go sur MADOC et testez-le.

```go
package main

import (
    "fmt"
    "math"
)

// Interface
type Forme interface {
    Perimetre() float64
}

func PlusGrandPerimetre(f1, f2 Forme) bool {
    return f1.Perimetre() > f2.Perimetre()
}

// Implantation pour les carrés
type Carre struct {
    Cote float64
}

func (c Carre) Perimetre() float64 {
    return 4 * c.Cote
}

// Implantation pour les cercles
type Cercle struct {
    Rayon float64
}

func (c Cercle) Perimetre() float64 {
    return 2 * math.Pi * c.Rayon
}

func main() {
    var ca Carre = Carre{Cote: 5}
    var ce Cercle = Cercle{Rayon: 2.5}
    fmt.Println(PlusGrandPerimetre(ca, ce))
}
```

Programme 1 – interface.go

Exercice 4.2.1
Implantez l'interface Forme une troisième fois par un type Rectangle qui représente des rectangles. Testez votre implantation.

Exercice 4.2.2
Sur le modèle de Perimetre, ajoutez une méthode Aire dans les méthodes demandées par l'interface Forme ainsi qu'une fonction PlusGrandeAire qui l'utilise. Mettre à jour les implantations de Carre, Cercle et Rectangle pour prendre en compte cette nouvelle méthode. Testez.