LA PROGRAMMATION ORIENTÉE OBJET DANS LE LANGAGE DE PROGRAMMATION GO

Go est-il un langage orienté objet ?

Promesse faite, promesse tenue, je vais vous parler de la POO (programmation orientée objet) en Go.

Voici d'abord ce qu'on peut retrouver dans la faq de go à propos de la POO.

Is Go an object-oriented language?

> Yes and no. Although Go has types and methods and allows an object-oriented style of programming, there is no type hierarchy. The concept of “interface” in Go provides a different approach that we believe is easy to use and in some ways more general. There are also ways to embed types in other types to provide something analogous—but not identical—to subclassing. Moreover, methods in Go are more general than in C++ or Java: they can be defined for any sort of data, even built-in types such as plain, “unboxed” integers. They are not restricted to structs (classes).
> *faq de go*

En gros il nous explique qu'il n'existe pas de hiérarchie des types en Go (par exemple la notion d'**héritage**). Mais qu'il existe des approches différentes de hiérarchisation de nos classes en utilisant par exemple les **packages imbriqués** où le répertoire racine sera la classe mère et les sous dossier des classes filles qui qui héritent de la classe mere.

Pour résumé Go n'est pas un langage de programmation purement orienté objet . Mais en combinant les structures et les packages, il est réalisable de se rapprocher le plus possible de la POO.

Création de nos classes

On va reprendre le même exemple vu dans le chapitre des structures, à savoir une structure `Personnage` avec les **attributs** suivants :

- la vie du personnage
- la puissance du personnage
- le nom du personnage
- Savoir si le personnage est mort ou pas
- L'inventaire du personnage

Et avec les méthodes suivantes :

- Presentation()
- ViePerdu()
- EstMort()

Nous allons créer notre structure `Personnage` dans un package personnage, ce qui nous donnera l'arborescence suivante :

```
$GOPATH/src
|___ personnage
| |___ personnage.go
main.go
```

Voici à quoi va ressembler notre classe `Personnage` dans le fichier **personnage.go** :

```
package personnage

import (
    "fmt"
)

type Personnage struct {
    Nom        string
    Vie        int
    Puissance  int
    Mort       bool
    Inventaire [3]string
}

/*
Affiche des informations sur un personnage

@return: void
*/
func (p Personnage) Affichage() { // déclaration de ma méthode Affichage() liée à ma
    fmt.Println("-------------------------------------------------")
    fmt.Println("Vie du personnage", p.Nom, ":", p.Vie)
    fmt.Println("Puissance du personnage", p.Nom, ":", p.Puissance)

    if p.Mort {
        fmt.Println("Vie du personnage", p.Nom, "est mort")
    } else {
        fmt.Println("Vie du personnage", p.Nom, "est vivant")
    }

    fmt.Println("\nLe personnage", p.Nom, "possède dans son inventaire :", p.Vie)

    for _, item := range p.Inventaire {
        fmt.Println("-", item)
    }
}
```

Rappel

Go exporte une variable/fonctions dans un autre package si et seulement si le nom commence par une lettre majuscule. Toutes les autres attributs/fonctions ne commençant pas par une lettre majuscule elles sont considérées comme **variables privées** ou **fonctions privées** par le paquet, c'est-à-dire qu'elles ne sont

pas exploitables en dehors du package où elles ont été initialisées.

Ensuite voici à quoi va ressembler notre fichier **main.go**

```
package main

import (
 "personnage"
)

func main() {

 magicien := personnage.Personnage{ // Instanciation de la classe Personnage
  Nom:        "magix",
  Vie:        100,
  Puissance:  20,
  Mort:       false,
  Inventaire: [3]string{"potions", "poisons", "bâton"},
 }

 magicien.Affichage()
}
```

Ici on importe le package `personnage` et sa structure `Personnage` qui pour nous représente une classe.

Une fois notre package importé, on crée ensuite une instance de notre classe qu'on nomme magicien et on lui affecte des valeurs par défaut.

À la fin de notre programme on invoque la méthode de notre classe Personnage, ce qui nous donne comme résultat :

```
--------------------------------------------------
Vie du personnage magix : 100
Puissance du personnage magix : 20
Vie du personnage magix est vivant

Le personnage magix possède dans son inventaire : 100
- potions
- poisons
- bâton
```

Les constructeurs

Le rôle du constructeur est de déclarer et d'initialiser les attributs d'une classe. Go ne supporte pas par défaut les constructeurs, mais il est possible de les créer en bidouillant avec les méthodes.

Avant de vous montrer comment on créée une sorte de constructeur en Go, voici d'abord une comparaison avec le le langage Java.

Voilà à quoi va ressembler par exemple notre structure Personnage en java:

```
public class Personnage{
        public String Nom;
        public int Vie;
        public int Puissance;
        public boolean Mort;
        String tableauChaine[] = {"rien", "rien", "rien"};
}
```

Voici comment on instancie une classe en java :

```
public class Main
{
    public static void main(String[] args)
    {
 // Instanciation de la classe Personnage
    Personnage magicien = new Personnage("magix", 100, 20, false, new String[] {"poti
    }
}
```

C'est grâce à l'opérateur `new`, qu'on arrive à faire appel à un constructeur d'une classe en java. Sur Go on va bidouiller un peu dans notre package personnage pour recréer l'opérateur `new` utilisé dans java (mais aussi dans beaucoup d'autres langages de programmations).

Pour ça on va créer une méthode nommée `New` dans package `personnage`, comme suit :

```
package personnage

import (
    "fmt"
)

type Personnage struct {
    Nom        string
    Vie        int
    Puissance  int
    Mort       bool
    Inventaire [3]string
}

/*
Créer une instance de la classe Personnage

@return: struct Personnage
*/
func New(Nom string, Vie int, Puissance int, Mort bool, Inventaire [3]string) Personn
    personnage := Personnage{Nom, Vie, Puissance, Mort, Inventaire}
    return personnage
}

/*
Affiche des informations sur un personnage

@return: void
*/
func (p Personnage) Affichage() { // déclaration de ma méthode Affichage() liée à ma
    fmt.Println("-------------------------------------------------")
    fmt.Println("Vie du personnage", p.Nom, ":", p.Vie)
    fmt.Println("Puissance du personnage", p.Nom, ":", p.Puissance)

    if p.Mort {
        fmt.Println("Vie du personnage", p.Nom, "est mort")
    } else {
        fmt.Println("Vie du personnage", p.Nom, "est vivant")
    }

    fmt.Println("\nLe personnage", p.Nom, "possède dans son inventaire :", p.Vie)

    for _, item := range p.Inventaire {
        fmt.Println("-", item)
    }
}
```

Notre fichier **main.go** va ressembler à ça :

```
package main

import (
 "personnage"
)

func main() {
 magicien := personnage.New("magix", 100, 20, false, [3]string{"potions", "poisons",
 magicien.Affichage()
}
```

Conclusion

Vous l'aurez sans doute compris même si Go ne prend pas en charge les classes, il reste tout de même possible d'utiliser les structures à la place des classes. Grâce à cette **fusion entre structures et packages** il est possible d'encore mieux **organiser son code** en s'orientant plus vers la POO. Mon but sur ce chapitre était de vous introduire à la POO. Il est bien sûr tout à fait possible d'aller encore plus loin en créant par exemple une notion d'héritage grâce aux interfaces, je créerai sûrement un article à propos de ce sujet.

Suggestion d'exercice

Avant de nous quitter sur ce chapitre je vous conseil de recréer le TP sur les morpions avec des classes ! Vous avez largement les ressources nécessaires pour réussir à bien cet exercice.