III.3 Types abstraits en `Swift` : `Protocol`

Les *protocoles* `Swift` fournissent un moyen élégant pour définir un type abstrait.

Un type concret implémentant un type abstrait définit par un protocole devra être conforme aux propriétés, fonctions de requête et de modification définies par le protocole.

La syntaxe des *protocoles* pour définir un type abstrait est la suivante :

```
protocol nom_type{

  // définition des propriétés et fonctions

}
```

Définition des fonctions de création d'un type abstrait

Pour définir des fonctions de création d'une instance d'un type abstrait, il faut donner la signature de(s) fonction(s) `init()`

À la création d'une instance, est appelé la fonction `init()` correspondante.

```
protocol NomProtocol {
  init()
  init(p1: Type1, p2: Type2)
}
// on suppose que NomType est conforme au protocole NomProtocol

// utilisation des fonctions de création:
var v : NomProtocol = NomType()
var x : NomProtocol = NomType(p1: v1, p2: v2)
```

Définition des propriétés d'un type abstrait

Un protocole ne précise pas si la *propriété* doit être une *propriété stockée* ou une *propriété calculée*. Il spécifie seulement le nom et le type de la propriété.

Le protocole précise également si la propriété est en lecture seule ou si elle peut être également modifiée.

```
protocol NomProtocol {
  var propriétéModifiable: Type { get set }
  var propriétéLectureSeule: Type { get }
}
// on suppose que NomType est conforme au protocole NomProtocol
// utilisation :
var v : NomProtocol = NomType()
v.propriétéModifiable = valeur
var x : Type = v.propriétéModifiable
var y : Type = v.propriétéLectureSeule
```

Définition des fonctions de requête d'un type abstrait

Pour définir des fonctions de *requête* d'un type abstrait, il suffit de donner la *signature* de la fonction

On appelle *signature* d'une fonction, son nom, son type et le type des paramètres.

```
protocol NomProtocol {
  // il y a forcément des paramètres puisque c'est une requête
  func requeteParam(p1: Type1, p2: Type2) -> Type3
}
// on suppose que NomType est conforme au protocole NomProtocol
// utilisation :
var v : NomProtocol = NomType()
var x : Type3 = v.requeteParam(p1: val1, p2: val2)
```

Définition des fonctions de modification d'un type abstrait

Pour définir des fonctions de *modification* d'un type abstrait, il faut donner la signature de la fonction mais aussi indiquer que celle-ci est *susceptible de modifier* l'instance

Cette indication se faire par le mot clef *mutating*.

```
protocol NomProtocol {
  // Self indique que la fonction doit renvoyer le même type concret
  // conforme à NomProtocol
  mutating func modif(p1: Type1, p2: Type2) -> Self
}
// on suppose que NomType est conforme au protocole NomProtocol
// utilisation :
var v : NomProtocol = NomType()
v.modif(p1: val1, p2: val2)
```

Exercice : Bataille navale - version 1

L'objectif est de programmer une bataille navale.

On nous demande d'écrire un programme qui permet de jouer à la bataille navale suivant les règles suivantes :

- 5 bateaux de tailles 1 à 4, dont deux de taille 3, sont stockés à des positions prédéterminées et immuables ;
- chaque bateau occupe des cases consécutives sur la même ligne ou même colonne d'une grille 20x20
- les lignes et les colonnes sont numérotées de 0 à 19 et une position est indiquée par les coordonnées (col, lig)
- le programme doit demander à l'utilisateur une position où tirer tant que tous les bateaux ne sont pas coulés

- à chaque proposition, l'algorithme répond :
  - « *touché* » si la position est occupé par un bateau et qu'il n'a pas été encore touché à cette position
  - « *coulé* » si la position est occupé par un bateau et que c'était la dernière position du bateau non encore touchée
  - « *en vue* » si la position n'est pas occupée par un bateau ou qu'elle correspond à une position déjà touchée, et que sur la ligne ou la colonne (ou les deux) se trouve une position non touchée occupée par un bateau
  - « *à l'eau* » dans les autres cas
- le programme s'arrête quand tous les bateaux ont été coulés

Exercice

Définissez les types nécessaires à la conception de ce programme