Programmation orientée objet en C#

Point sur le laboratoire 2

Point sur le labo 2

- Propriétés
- Paramètres de méthodes

PROPRIÉTÉS

Comme s’il s’agissait d’attributs publics… en apparence.

Propriétés

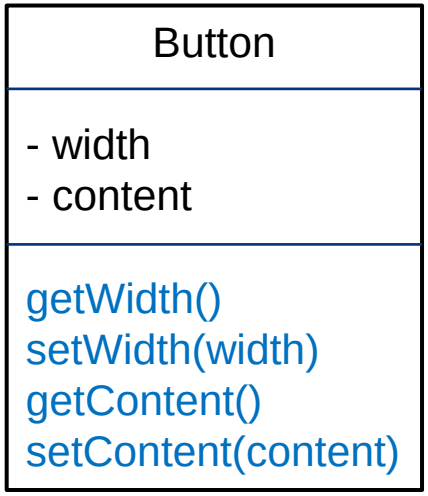
- Principe d'**encapsulation** (un des piliers de l'OO) :
  - État (attributs) + Comportement (méthodes) groupés en classes
  - Partie visible et partie cachée/protégée
  - Accès indirects pour permettre un couplage plus faible

- En Java, cela se traduit par des **getters/setters**.

- **Remarque** : rien n'oblige les getters/setters à suivre la structure des attributs !
  - *Exemple* : on stocke une date dans un attribut unique mais on offre des getters/setters séparés pour le jour, le mois et l'an.
  - La structure interne peut changer sans qu'on ne doive modifier les getters/setters utilisables depuis l'extérieur !

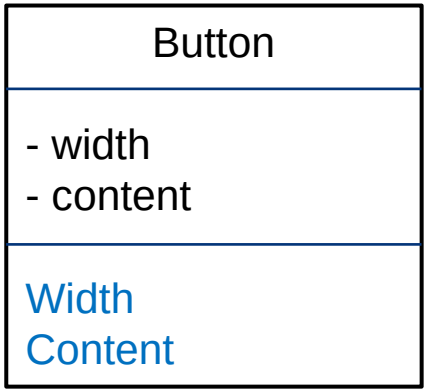
Propriétés

En Java… getters/setters




```
myButton.setWidth("100px");
String txt;
txt = myButton.getContent();
```

En C#... propriétés




```
myButton.Width = "100px";
string txt;
txt = myButton.Content;
```

Attributs privés

Accès depuis l'extérieur

Propriétés

- Caractéristiques des propriétés (en C#) :
  - Syntaxe des attributs publics
    ```
    myButton.Width = "300 px";
    myButton.Content = "Lancer la partie";
    ```
  - Mais il s'agit en fait de **méthodes** !
    (ce qui explique la majuscule au début [convention])
  - Utilisation en lecture → appel au getter de la propriété
    Utilisation en écriture → appel au setter de la propriété
    ```
    myButton.Content += "!"; // deux appels !
    ```
  - Malgré la notation, peut cacher un code plus complexe !

Propriétés

- Getters/setters vs propriétés ?
  - **Efficacité / puissance** : aucune différence !
  - **Syntaxe** : on évite les () avec les propriétés.
  - **Lisibilité** : on cache ce qui peut être un code complexe !
  - « **Pragma** » : c'est la convention en C#

- Note : les propriétés existent également en Javascript et en PHP.

Propriétés

- **Syntaxe (de base)**

```
private string firstName;

public string FirstName
{
        get
        {
          return firstName;
        }
        set
        {
          firstName = value;
        }
}
```

Ici, un exemple lié à un attribut privé de nom similaire et de même type, mais ce n'est pas obligatoire !

Ici, getter/setter sans vérification ni code particulier… mais on pourrait y trouver (quasiment) n'importe quel code !
Et on peut faire évoluer le code !

Dans le setter, « value » est le nom donné par défaut à la valeur qu'on tente d'attribuer à la propriété.

Propriétés

- **Propriété auto-implémentée**
  - Déclare automatiquement l'attribut privé
  - Implémente automatiquement un getter et un setter standard

Exemple de base

```
public string Login { get; set; }
```

Un attribut privé est créé automatiquement mais on n'y a pas d'accès direct !

Exemple avec initialisation

```
public string City { get; set; } = "Namur";
```

Exemple avec implémentation partielle (accès uniquement en lecture)

```
public string City { get; } = "Namur";
```

Note : il existe d'autres syntaxes non présentées ici !

Exemple avec visibilités différentes

```
public string Color { get; private set; }
```

PARAMÈTRES DES MÉTHODES

Au-delà du passage par valeur…

Paramètres des méthodes

- **Passage standard** (cfr notes précédentes) :
  - Types-valeurs : passage par **valeur**
    La méthode travaille sur une copie.
  - Types-références : passage par **adresse**
    (= passage par valeur de la référence)
    La méthode travaille sur l'objet lui-même.

- Forcer un passage par adresse/résultat via **ref** ou **out** :
  - `CalculerDégâts (ref int pointsDeVie)`
    La variable utilisée comme paramètre effectif doit être initialisée.
    La méthode peut la modifier directement.
  - `EffectuerVoyage (out int consommation, out int pollution)`
    La variable utilisée comme paramètre effectif doit être déclarée.
    La méthode lui affecte une valeur.

Paramètres des méthodes

- Paramètres **optionnels**

```
public decimal Intérêts (decimal capital, decimal taux = 3)
```

- Paramètres **en nombre variable**

```
public int Sum (params int[] values)
```

Dans la méthode, values fonctionne comme un tableau.

- Paramètres **nommés** (permet de donner les paramètres dans un ordre différent lors de l'appel)

```
IntérêtsCompte (taux : 2.5, capital : 500)
```