



Cégep de Saint-Hyacinthe  
Département d'informatique

Programmation orientée objet

420-2DP-HY

**(3-3-3)**

## **Création et utilisation de bibliothèques**

(Version 1.2)

### **1.5 à 3 heures**

Préparé par

**Martin Lalancette**

Comprendre les éléments suivants :

- Encapsulation d'objets
- Création et utilisation d'une bibliothèque
- Les références
- Gestion d'une solution multiprojets

## Table des matières

Introduction.....	3
Encapsulation d'objets .....	3
Création d'un projet de type Bibliothèque de classes .....	4
Les références ou dépendances .....	6
Bibliographie.....	9

## Introduction

Cette séquence a pour but de vous initier aux notions de base nécessaires à la programmation à base d'objets et d'événements. Nous commencerons par énoncer les éléments théoriques appuyés d'exemples simples et faciles à reproduire. Afin d'axer l'attention sur la compréhension de ces notions, il y aura des exercices à faire tout au long de cette séquence. **La séquence portera sur la création et l'utilisation d'une bibliothèque.**



**Exercice 1. :** S'assurer d'avoir créé l'arborescence ici haut mentionnée sur votre C ou votre clé USB.

## Encapsulation d'objets

Un petit rappel, lorsque nous faisons appelle à une méthode d'un objet et que nous obtenions un résultat sans connaître la complexité de cet objet ou méthode, ce concept qui cache cette complexité s'appelle l'**encapsulation**. En d'autres termes, de l'extérieur, les objets rendent visible ce qui est utile, et dissimulent ce qui ne l'est pas. Ceci est pour des raisons pratiques, sécuritaires, et aussi confidentielles.

Quelques caractéristiques :

- L'encapsulation est une caractéristique de la POO qui rend inaccessible le contenu privé d'un objet.
- Ce qui est privé est donc isolé de l'extérieur et demeure loin des manipulations malveillantes, maladroites, ou inappropriées par ceux qui utilisent les objets.
- Il est alors concevable de garder le contrôle sur l'intégrité, sur le comportement et sur l'évolution des objets.

Les efforts peuvent être concentrés sur ce qui est public, en ajoutant de la validation là où c'est le plus à risque d'engendrer des erreurs.

Une bonne pratique, toujours reliée à l'encapsulation, consiste également à isoler nos classes dans des bibliothèques afin de rendre les fonctionnalités disponibles à divers projets. C'est ce que nous allons voir dans cette séquence. Les avantages :

- Fonctionnalités réutilisables
- Centralisation du code (Bonne architecture)
- Applicable dans divers type de projet (WPF, Windows Forms, console, Web)

Voici un nouveau modificateur d'accès :

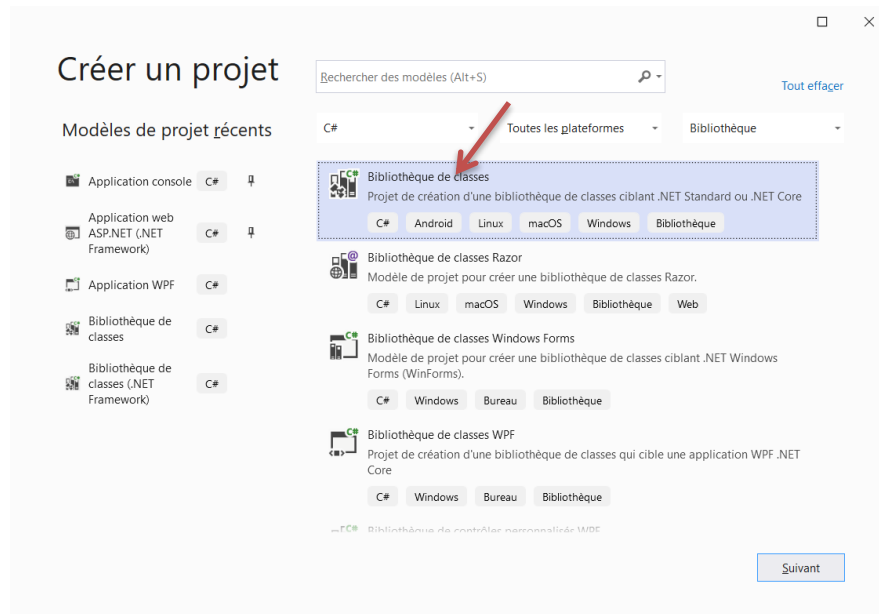
Modificateur	Signification
<b>private</b>	Seul le code de la classe elle-même peut avoir accès aux membres. Invisible de l'extérieur.
<b>public</b>	Tout autre code du même assembly (c.-à-d. : programme ou une librairie) ou non, qui fait référence à l'objet peut accéder au membre.
<b>internal</b>	Tout code du même assembly (dll ou exe) peut accéder au membre. Les autres <i>assemblies</i> n'ont pas accès.

## Création d'un projet de type Bibliothèque de classes

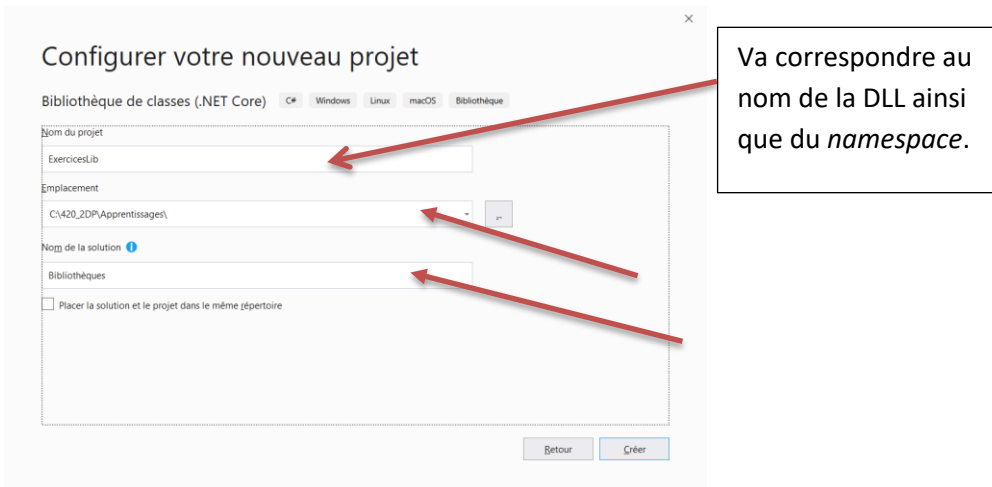
Une bibliothèque de classes est la traduction de *library* (en anglais) qui se définit comme une collection de classes (fonctionnalités), d'interfaces et/ou des types qui, une fois compilée, en résulte un fichier .DLL (*Dynamic Link Library*) ou communément appelée *assembly* en C#.

Dans cette section nous allons apprendre comment créer une bibliothèque de classes que nous utiliserons dans deux types de projets différents (Console et WPF). Voici les étapes à suivre pour créer une bibliothèque de classes :

1. Ouvrir la solution et la sélectionner,
2. Cliquer droit avec la souris et choisir **Nouveau/Projet...**
3. Choisir **Bibliothèque de classes (.NET Core)**. Exemple :



4. Sélectionner les informations et donner le nom à la bibliothèque.



5. Cliquer sur **Créer**.

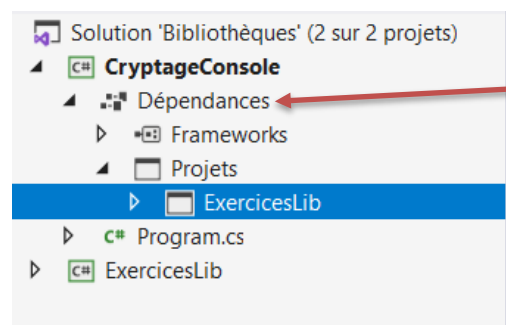
**Exercice 2. :** Effectuer la procédure ci-haute pour créer la solution **Bibliothèques** et la bibliothèque nommée **ExercicesLib**. Une classe nommée **Class1** sera créée automatiquement. Renommez-la **STHCryptage**.

C'est dans cette bibliothèque que nous allons coder les algorithmes des prochains exercices.

**Exercice 3. :** Ajouter dans la classe **STHCryptage**, deux méthodes **Encrypter** et **Décrypter** qui prennent en paramètre une phrase. Une des plus anciennes méthodes de cryptage était de décaler de 1 la position des lettres composant la phrase. Exemple : A devient B, G devient H, etc.

## Les références ou dépendances

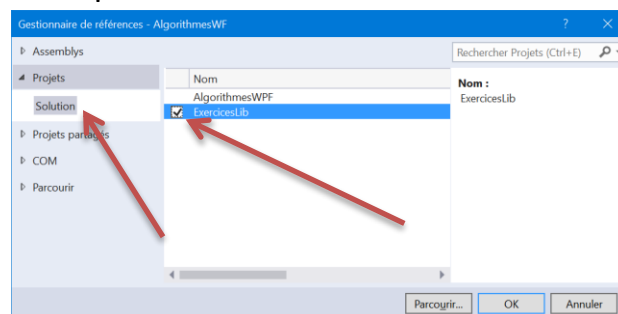
Afin de pouvoir avoir accès à une bibliothèque dans un projet, nous devons l'ajouter dans les références de ce projet.



Donc voici la procédure à suivre afin d'inclure cette nouvelle bibliothèque :

1. Générer la DLL en compilant le projet de cette bibliothèque
2. Choisir le projet dans lequel vous souhaitez l'inclure, cliquer droit sur **Références** et choisir **Ajouter une référence....**
3. Choisir Projets (à gauche) et cocher le nom de la bibliothèque.

Exemple :



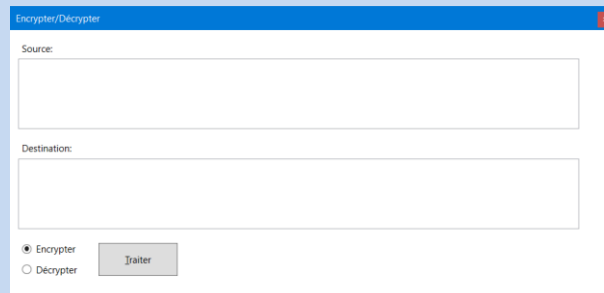
4. Cliquer sur le bouton OK.

**Exercice 4. :** Ajouter un nouveau projet nommé **CryptageConsole**. Ce programme devra demander au démarrage quelle opération est souhaitée par l'utilisateur (soit E pour Encrypter ou D pour Décrypter). Poser les questions en fonction de ce choix et afficher la réponse en utilisant les méthodes de la classe **STHCryptage** de la bibliothèque précédente. Vous devez ajouter une référence et créer un objet.

À partir de maintenant, les membres publiques de la classe **STHCryptage** seront accessibles dans vos projets. Vous devez utiliser le nom du *namespace* pour y accéder, sinon vous pouvez ajouter :

```
using ExercicesLib;
```

**Exercice 5. :** Ajouter un nouveau projet de type WPF (.NET Core) nommé **CryptageWPF**. Concevoir l'interface suivant :



Ajouter le code dans le bouton **Traiter** afin d'appeler les méthodes de la classe **STHCryptage**. Ajouter une référence.

**Exercice 6. :** Ajouter dans la classe **STHCryptage** une propriété nommée **Clé**. Modifier les méthodes **Encrypter** et **Décrypter** pour qu'elles prennent en paramètre une phrase et que si la clé est vide alors la méthode précédente est appliquée, sinon appliquer la nouvelle technique. Cette fois-ci, il faut utiliser une autre technique de cryptage → substitution aléatoire. Pour cela, on utilise un alphabet-clé, dans lequel les lettres se succèdent de manière désordonnée, par exemple : **HY-LUJPVREAKBNDOFSQZCWMGITX**

C'est cette clé qui va servir ensuite à coder le message. Selon notre exemple, les A deviendront des H, les B des Y, les C des L, etc. Programmer le tout et faire fonctionner dans les autres projets : Poser une question supplémentaire et ajouter des champs visuels, si nécessaire.

**Exercice 7. :** Ajouter dans la classe **Exercices**, une méthode **GénérerCléAléatoire** qui consiste à produire une clé de 26 caractères contenant toutes les lettres de l'alphabet mélangées. Il ne peut y avoir deux fois la même lettre (exemple de l'exercice précédent). Programmer le tout et faire fonctionner dans les autres projets.



## Bibliographie

**Aucune source spécifiée dans le document actif.**