

Programmation Orientée Objet C#

Cas Navire

Objectifs du TP :

- ✓ S'initier à la programmation objet en C#
- ✓ Utiliser les bonnes pratiques pour le débogage
- ✓ Lire une documentation et savoir l'appliquer
- ✓ Réviser les concepts algorithmiques
- ✓ Se forcer à se poser des questions et en obtenir des réponses

Ce TP n'a pas vocation à être un TP professionnel. Il est seulement pédagogique et a pour but de vous faire faire vos premiers pas en POO

Prérequis :

- ✓ Bases algorithmiques
- ✓ Bases du langage C#
- ✓ Bases théoriques de la POO
- ✓ Principales fonctionnalités de l'IDE Eclipse

Dans ce TP, vous serez tour à tour :

- ✓ Concepteur de classe :

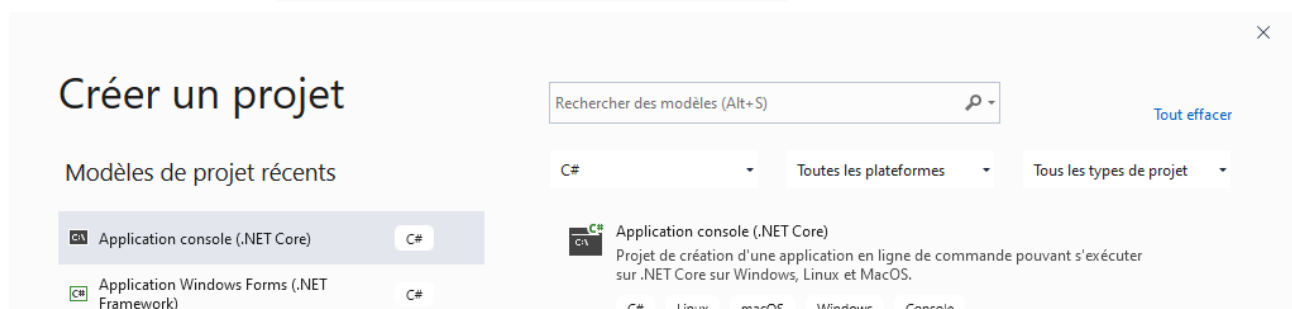
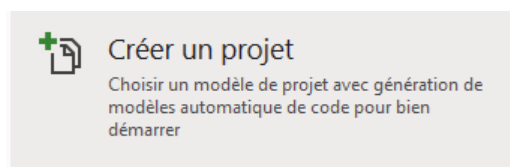


- ✓ Utilisateur de classe :



Partie 1 : CREATION DU PROJET

Vous allez créer un projet Console (.Net Core) appelé TP1Navire:



Configurer votre nouveau projet

Bibliothèque de classes (.NET Standard) C# Android iOS Linux macOS Windows Bibliothèque

Nom du projet

TP1Navire

Emplacement

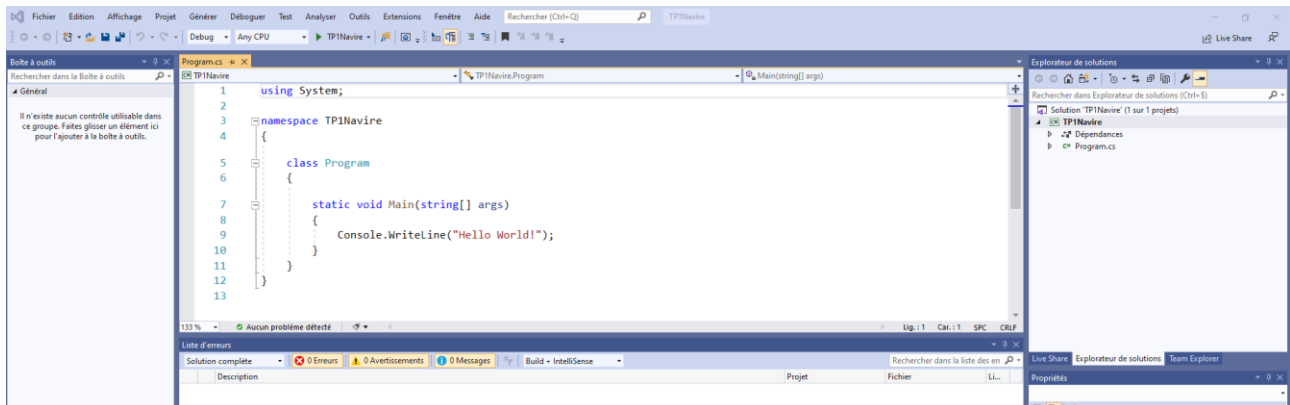
T:\lycee\2019-2020\SIO1\SLAM2\TP1_C#OO\

Nom de la solution

TP1Navire

☐ Placer la solution et le projet dans le même répertoire

Vous devriez obtenir ceci :



Partie 2 : AVANT DE COMMENCER A PROGRAMMER

Je vous invite, au cours de ce TP à utiliser les fonctions de débogage de Visual Studio 2019.

Vous serez en mesure :

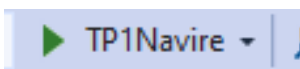
- ✓ De poser un point d'arrêt (break point)
- ✓ D'exécuter le programme pas à pas (Step into (F5), Step over (F6), Step return (F7), Resume (F8))
- ✓ D'être en mesure d'analyser l'état des variables sur un point d'arrêt

A voir :

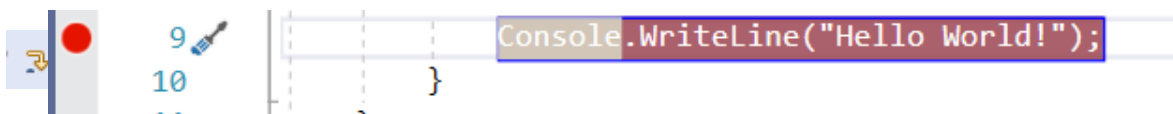
<https://docs.microsoft.com/fr-fr/visualstudio/debugger/navigating-through-code-with-the-debugger?view=vs-2019>



Rappels :



Début du débogage. Il suffit de mettre un point d'arrêt dans la marge pour que le mode debug commence automatiquement



Pas à pas détaillé (F11) : Exécution pas à pas détaillé y compris dans les modules appelés.



Pas à pas principal (F10) : exécution pas à pas sans entrer dans les modules appelés que vous considèrerez comme valides



Pas à pas sortant (Maj F11) : termine l'exécution du module appelé et va à l'instruction qui a fait appel au module.

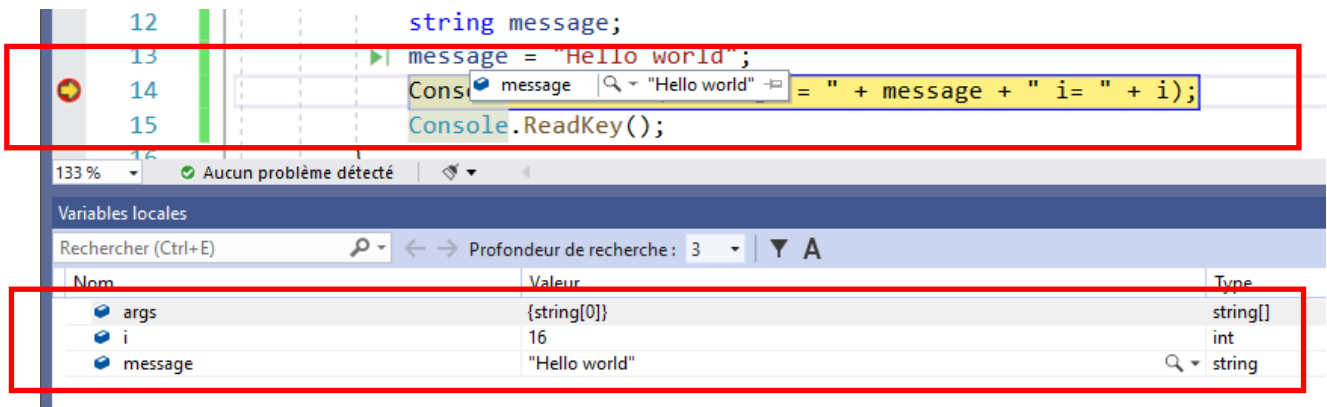


Resume (F8) : Poursuit l'exécution du programme jusqu'au point d'arrêt suivant ou la fin du programme.



Arrêter le débogage (MAJ + F5)

Visualisation des variables :



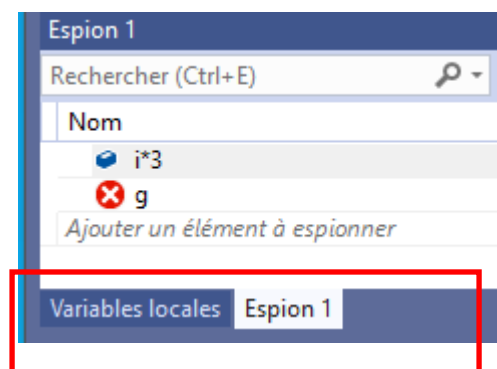
The screenshot shows a Java program with the following code:

```
12 string message;
13 message = "Hello world";
14 Console.WriteLine("Hello world" + " i = " + i);
15 Console.ReadKey();
```

The local variables window is open, showing the following variables:

Nom	Valeur	Type
args	{string[0]}	string[]
i	16	int
message	"Hello world"	string

Vous pouvez aussi créer vos propres expressions ou rajouter des variables espion :
Vous saisissez une expression, votre IDE vous affiche sa valeur.



The 'Espion 1' window shows a search bar with the text 'Rechercher (Ctrl+E)'. Below the search bar, there is a list of variables being monitored:

- ☒ i*3
- ☒ g

Below the list, there is a button labeled 'Ajouter un élément à espionner'. At the bottom of the window, there is a tab labeled 'Variables locales' and a sub-tab labeled 'Espion 1'.



Modifiez les valeurs de message et de i dans la fenêtre des variables ou la fenêtre des espions et poursuivez le programme.

Partie 3 : LA CLASSE NAVIRE

On vous propose de créer votre première classe. C'est la classe Navire, dont voici une représentation UML.



- ✓ imo : numéro d'identification des navires au niveau international. Il est composé des trois lettres "IMO", suivies du numéro à sept chiffres. Ce numéro ne peut être modifié.
Vous pourrez trouver des numéros IMO sur le site : <http://www.vesselmarinefinder.com>
- ✓ nom : nom du navire, il peut être modifié au cours de la vie du navire
- ✓ libelleFret : nature de la cargaison (hydrocarbure, containers,...), elle peut être modifiée au cours de la vie du navire.
- ✓ qteFretMaxi : quantité maximale de fret que peut embarquer un navire. Exprimé en tonnes entières, ...), elle peut être modifiée au cours de la vie du navire.



Le constructeur est **surchargé**. Lors de l'instanciation, on peut fournir toutes les valeurs des attributs ou seulement le numéro IMO et le nom. Les propriétés libelleFret et qteFetmaxi sont alors initialisées respectivement à *Indéfini* et *0*.

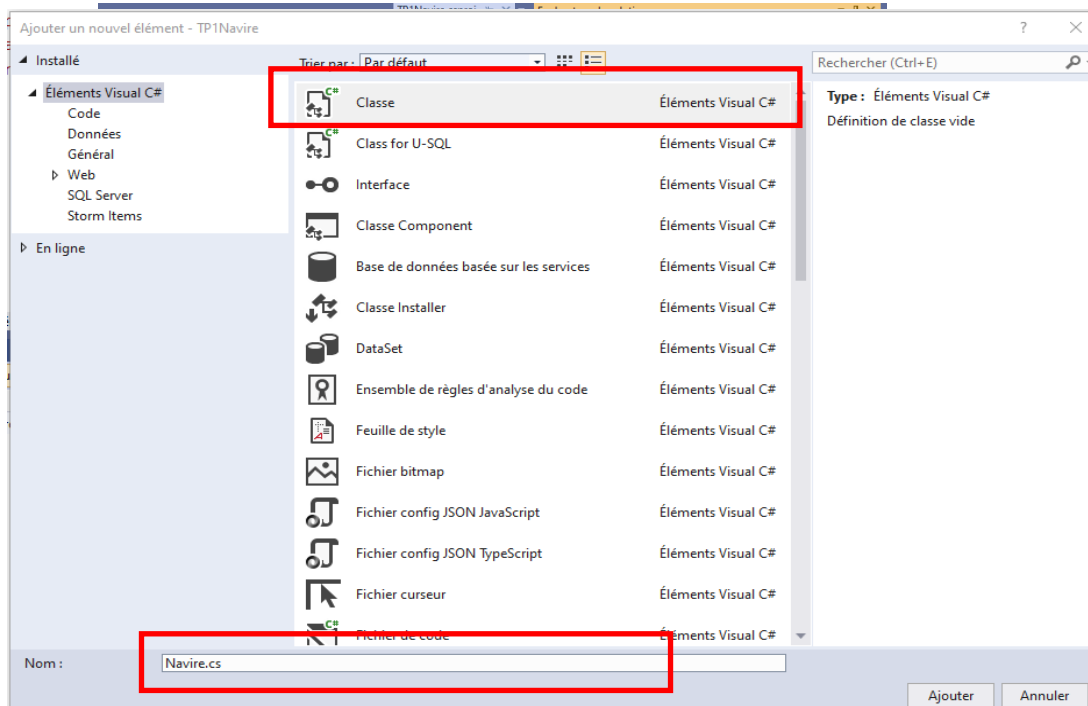
Dans cette partie, vous serez :

Classes		
Navire		

Partie 4 : PREMIERE CLASSE EN C#

Vous allez maintenant créer la classe Navire

Pour créer une classe :



Attention à bien saisir l'extension .cs pour le nom de la classe (qui sera aussi le nom du fichier contenant le code de la classe).

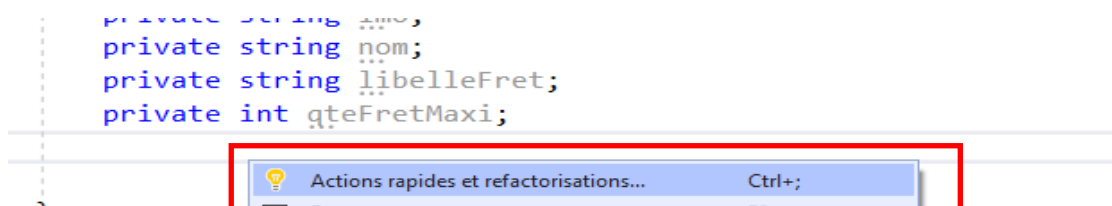


Avant de créer votre classe sachez que vous pouvez générer automatiquement :

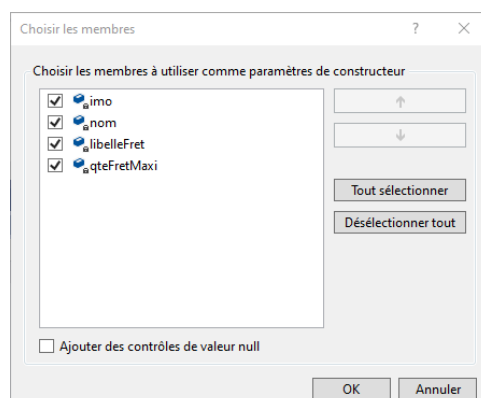
- ✓ Le constructeur
- ✓ Les getters et setters

Le constructeur :

Click bouton droit à l'endroit où vous voulez insérer le constructeur :



Puis : Générer le constructeur :

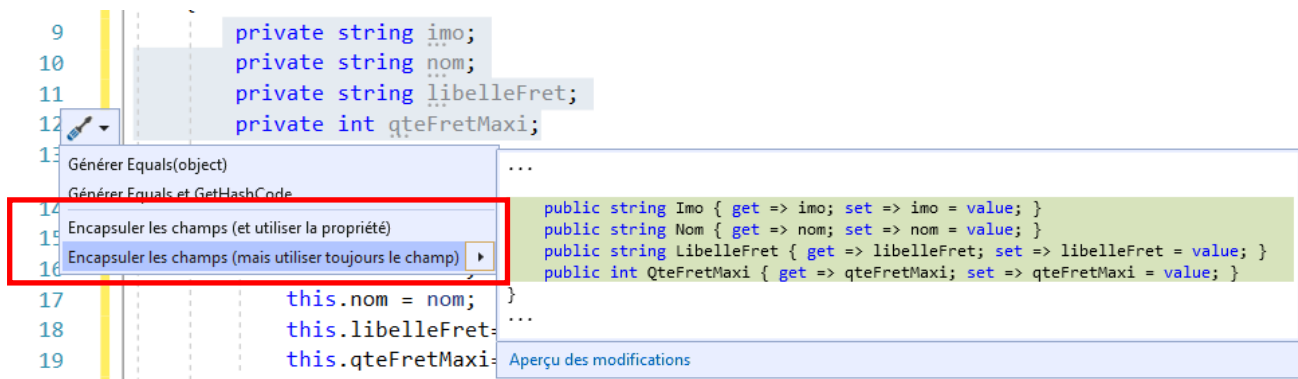


Vous choisirez quels attributs à valoriser vous passerez en paramètre au constructeur.
Et votre constructeur est créé:

```
public Navire(string imo, string nom, string libelleFret, int qteFretMaxi)
{
    this.imo = imo;
    this.nom = nom;
    this.libelleFret=libelleFret;
    this.qteFretMaxi=qteFretMaxi;
}
```

Pour les Properties (getters et setters) :

Sélectionnez les champs pour lesquels vous voulez créer les propriétés, click bouton droit /
Actions rapides et refactorisations :



Vous aurez le choix :

- ✓ D'utiliser la propriété (property) dans le constructeur,
- ✓ D'utiliser le champ dans le constructeur.



Vous pouvez mixer les 2 méthodes en fonction de ce que vous voulez faire.
Notamment, vous utiliserez la propriété si vous devez effectuer un contrôle sur la valeur à attribuer au champ. Par exemple une valeur comprise entre 2 bornes.
Vous pourrez éventuellement rectifier par la suite.

Pour l'instant travaillons avec les champs.

```
0 références
public Navire(string imo, string nom, string libelleFret, int qteFretMaxi)
{
    this.imo = imo;
    this.nom = nom;
    this.libelleFret=libelleFret;
    this.qteFretMaxi=qteFretMaxi;
}

0 références
public string Imo { get => imo; set => imo = value; }
0 références
public string Nom { get => nom; set => nom = value; }
0 références
public string LibelleFret { get => libelleFret; set => libelleFret = value; }
0 références
public int QteFretMaxi { get => qteFretMaxi; set => qteFretMaxi = value; }
```

Vous organiserez proprement votre code de classe, pour retrouver dans l'ordre :

- ✓ Les attributs (privés)
- ✓ Le ou les constructeurs
- ✓ Les méthodes privées, puis protected puis publiques
- ✓ les properties.

Dans cette partie, vous serez :

Classes		
Navire		



Travail à faire

Créer la classe Navire. Attention, certains champs ne sont pas accessibles en modification (readonly) et le constructeur est surchargé et il n'y aura pas de duplication de code, il faudra que le deuxième constructeur fasse appel au premier.





Remarquez le nommage des attributs et des paramètres du code généré.
Soyez vigilants avec la casse

Partie 5 : UTILISATION DE LA CLASSE NAVIRE

Vous allez travailler dans la classe Program qui a été créée à la création du projet et qui contient la méthode Main(...).

Dans cette partie, vous serez :

Situation		
Vous écrivez du code dans la classe <i>Programme</i>	Navire	Navire
Vous écrivez du code dans la classe <i>Navire</i>	Navire	Navire



Travail à faire

Ecrire le code suivant de la classe Programme

Vous allez créer la méthode `TesterInstanciations()` qui va permettre d'instancier une série de navires. Cette méthode sera appelée depuis la méthode `Main()`.

```
static void Main()
{
    TesterInstanciations();

    Console.ReadKey();
}

public static void TesterInstanciations()
{
    // déclaration de l'objet unNavire de la classe Navire
    Navire unNavire;
    // Instantiation de l'objet
    unNavire = new Navire("IM09427639", "Copper Spirit", "Hydrocarbures", 156827);
    // Declaration ET instantiation d'un autre objet de la classe Navire
    Navire unAutreNavire = new Navire("IM09839272", "MSC Isabella", "Porte-conteneurs", 197500);
    // ??
    unAutreNavire = new Navire("IM08715871", "MSC PILAR");
}
```



Travail à faire

Créer la méthode `Affiche(Navire navire)` qui affiche les caractéristiques du navire passé en paramètre. Vous modifierez la méthode `testerInstanciations()`, voir ci-dessous.

Vous devriez obtenir le résultat ci-dessous :

```
Identification : IM09427639
Nom : Copper Spirit
Type de frêt : Hydrocarbures
Quantité de Frêt : 156827
-----
Identification : IM09839272
Nom : MSC Isabella
Type de frêt : Porte-conteneurs
Quantité de Frêt : 197500
-----
Identification : IM08715871
Nom : MSC PILAR
Type de frêt : Indéfini
Quantité de Frêt : 0
-----
```

```
public static void TesterInstanciations()
{
    // déclaration de l'objet unNavire de la classe Navire
    Navire unNavire;
    // Instantiation de l'objet
    unNavire = new Navire("IM09427639", "Copper Spirit", "Hydrocarbures", 156827);
    Affiche(unNavire);
    // Declaration ET instantiation d'un autre objet de la classe Navire
    Navire unAutreNavire = new Navire("IM09839272", "MSC Isabella", "Porte-conteneurs", 197500);
    Affiche(unAutreNavire);
    // ??
    unAutreNavire = new Navire("IM08715871", "MSC PILAR");
    Affiche(unAutreNavire);
}
```

Vous allez maintenant faire mieux et utiliser la méthode `ToString()`:

Voir : <https://docs.microsoft.com/fr-fr/dotnet/api/system.object.tostring?view=netframework-4.8>



La méthode `toString()` permet de sérialiser une instance d'une classe.
Elle est proposée dans la classe `Object`



Travail à faire

Cherchez le fonctionnement de cette méthode et modifiez le code de la classe `Navire` (méthode `ToString()`) et de la méthode `TesterInstanciations()`.

Vous devriez arriver au même résultat :

```

Identification : IM09427639
Nom : Copper Spirit
Type de frêt : Hydrocarbures
Quantité de Frêt : 156827
-----
Identification : IM09839272
Nom : MSC Isabella
Type de frêt : Porte-conteneurs
Quantité de Frêt : 197500
-----
Identification : IM08715871
Nom : MSC PILAR
Type de frêt : Indéfini
Quantité de Frêt : 0
-----

```

Partie 6 : CLASSE PORT

Vous allez maintenant créer la classe `Port`.



Données membres :

- ✓ `nom` : nom du port, chaîne de caractères non modifiable
- ✓ `nbNaviresMax` : nombre de bateaux maximum que le port peut accueillir à un instant `t`. Il est accessible en lecture et en modification. Il est initialisé à 5 dans la déclaration
- ✓ `navires` : objet de la classe `List` stockant des objets de la classe `Navire`. Représente les navires actuellement à quai dans le port.

Fonctions membres

- ✓ Constructeur : non surchargé, on lui fournit en paramètre le nom du port
- ✓ `public EnregistrerArrivee(Navire navire)` : méthode publique qui va stocker l'objet de la classe `Navire` passé en paramètre dans la collection `navires`. On ne peut rajouter un navire dans le port que si la capacité du port le permet. Si on essaye d'enregistrer un navire dans un port ayant atteint sa capacité maximale, on affiche le message :
Ajout impossible, le port est complet

- ✓ `public EnregistrerDepart(String imo) :` va retirer le navire dont le numéro IMO est égal au numéro IMO passé en paramètre. Elle doit au préalable vérifier si le navire est bien présent dans la collection navires.
- ✓ `public EstPresent(String imo) :` retourne un booléen. True si le navire passé en paramètre est présent dans la collection navires, false dans le cas contraire. **Le test de recherche se fera sur l'égalité des numéros IMO.**
- ✓ `private RecupPosition(String imo) :` retourne la position i du navire dans la collection navires dont le numéro IMO est égal au numéro IMO passé en paramètre ou -1 si le numéro IMO n'a pas été trouvé.
- ✓ `private RecupPosition(Navire navire) :` idem, mais la comparaison se fait sur l'objet navire passé en paramètre. **Vous utiliserez la méthode contains de la classe List pour effectuer votre recherche.**



Avant de commencer, étudier les méthodes de la classe List, ça peut vous aider
<https://bre.is/RjnKYa2f>

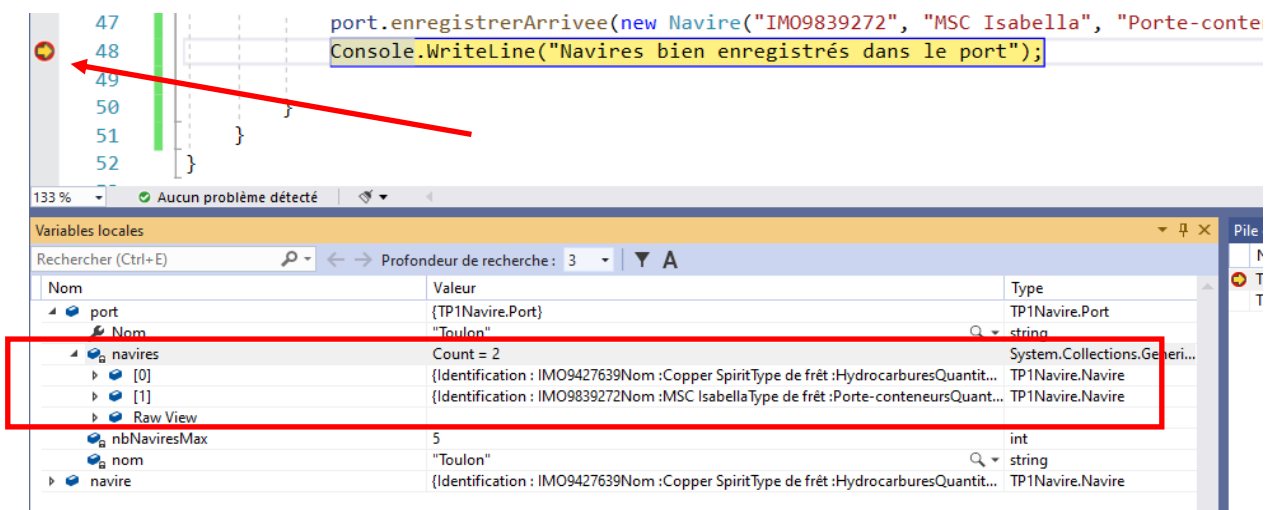


Travail à faire

Ecrire la classe Port et le code des méthodes.

Pour tester les méthodes, vous écrirez les méthodes suivantes dans la classe Programme :

- ✓ **TesterEnregistrerArrivee()** pour tester `Port::enregistrerArrivee(Navire navire) :`
 Vous mettrez les points d'arrêt pour contrôler que les navires ont bien été rajoutés à la collection navires



The screenshot shows a code editor with the following code snippet:

```

47
48 port.enregistrerArrivee(new Navire("IMO9839272", "MSC Isabella", "Porte-conteneurs"));
49 Console.WriteLine("Navires bien enregistrés dans le port");
50
51 }
52

```

A red arrow points from the code to the Variables Locales window. The Variables Locales window shows the following variables:

Nom	Valeur	Type
port	{TP1Navire.Port}	TP1Navire.Port
Nom	"Toulon"	string
navires	Count = 2	System.Collections.Generic.List<TP1Navire.Navire>
[0]	{Identification : IMO9427639Nom : Copper SpiritType de frêt : HydrocarburesQuantit...	TP1Navire.Navire
[1]	{Identification : IMO9839272Nom : MSC IsabellaType de frêt : Porte-conteneursQuant...	TP1Navire.Navire
Raw View		
nbNaviresMax	5	int
nom	"Toulon"	string
navire	{Identification : IMO9427639Nom : Copper SpiritType de frêt : HydrocarburesQuantit...	TP1Navire.Navire

Appel de la méthode dans la méthode main() :

Vous devriez obtenir ceci (dernière ligne de la procédure TesterEnregistrerArrivee:

```
Navires bien enregistrés dans le port
```



Travail à faire

Rajouter 4 navires dans le port et vous devriez obtenir ceci :

```
Ajout impossible, le port est complet
Navires bien enregistrés dans le port
```

Il y a là une incohérence... expliquez pourquoi.



Travail à faire

Au lieu d'afficher le message : *Ajout impossible, le port est complet*, il vous est demandé de déclencher une exception qui sera interceptée par la classe Programm et qui affichera le même message. Constatez la différence :

```
T:\lycee\2019-2020\SIO1\SLAM2\TP1_C#OO\TP1Navire\TP1
Ajout impossible, le port est complet
```

✓ **Port::TesterRecupPosition()** pour tester Port:: recupPosition(String imo)

Cette méthode sera écrite dans la classe Port

```
public void TesterRecupPosition()
{
    this.EnregistrerArrivee (new Navire("IM09427639", "Copper Spirit", "Hydrocarbures", 156827));
    this.EnregistrerArrivee(new Navire("IM09839272", "MSC Isabella", "Porte-conteneurs", 197500));
    this.EnregistrerArrivee(new Navire ("IM08715871", "MSC PILAR"));
    String imo = "IM09427639";
    Console.WriteLine("Indice du navire " + imo + " dans la collection : " + this.recupPosition(imo));
    imo = "IM08715871";
    Console.WriteLine("Indice du navire " + imo + " dans la collection : " + this.recupPosition(imo));
    imo = "IM01111111";
    Console.WriteLine("Indice du navire " + imo + " dans la collection : " + this.recupPosition(imo));
}
```

Je vous conseille de toujours tester le premier et le dernier élément dans la collection Pour limiter le risque de "plouf la "

Et dans la classe Programme :

```
public static void TesterRecupPosition()
{
    (new Port("Toulon")).TesterRecupPosition();
}
```

Dans la méthode main :

```
TesterRecupPosition();
```

Vous devriez obtenir le résultat suivant :

```
Indice du navire IM09427639 dans la collection : 0
Indice du navire IM08715871 dans la collection : 2
Indice du navire IM01111111 dans la collection : -1
```

✓ **Port::testerRecupPositionV2()** pour tester Port:: recupPositionV2(navire navire)

Cette méthode sera écrite dans la classe Port. On ne passe plus le numéro IMO en paramètre, mais un objet de la classe Navire.

```
public void TesterRecupPositionV2()
{
    Navire navire = new Navire("IM09427639", "Copper Spirit", "Hydrocarbures", 156827);
    this.EnregistrerArrivee(navire);
    this.EnregistrerArrivee(new Navire("IM09839272", "MSC Isabella", "Porte-conteneurs", 197500));
    this.EnregistrerArrivee(new Navire("IM08715871", "MSC PILAR"));
    Console.WriteLine("Indice du navire " + navire.Imo + " dans la collection : " + this.recupPosition(navire));
    Navire unAutreNavire = new Navire("IM08715871", "MSC PILAR");
    Console.WriteLine("Indice du navire " + unAutreNavire.Imo + " dans la collection : " + this.recupPosition(unAutreNavire));
    unAutreNavire = new Navire("IM08715871", "MSC PILAR", "Porte canteneurs", 52181);
    Console.WriteLine("Indice du navire " + unAutreNavire.Imo + " dans la collection : " + this.recupPosition(unAutreNavire));
}
```

Et dans la classe Programme :

```
public static void TesterRecupPositionV2()
{
    (new Port("Toulon")).TesterRecupPositionV2();
}
```

Dans la méthode main :

```
TesterRecupPositionV2();
```

Vous devriez obtenir le résultat suivant :

```
Indice du navire IM09427639 dans la collection : 0
Indice du navire IM08715871 dans la collection : -1
Indice du navire IM08715871 dans la collection : -1
```



Qu'en déduisez-vous ?

Quelle méthode utiliserez-vous pour faire le test de présence d'un navire dans le port ?

✓ **testerEnregistrerDepart()** de la classe Port

Cette méthode retire le navire dont le numéro IMO est passé en paramètre de la collection navires. Si le navire n'est pas dans le port, une exception sera levée et gérée dans la méthode Main().

Pour tester la méthode Port::enregistrerDepart(String imo), vous écrirez cette méthode dans la classe Programm.

```
public static void TesterEnregistrerDepart()
{
    Port port = new Port("Toulon");
    port.EnregistrerArrivee(new Navire("IM09427639", "Copper Spirit", "Hydrocarbures", 156827));
    port.EnregistrerArrivee(new Navire("IM09839272", "MSC Isabella", "Porte-conteneurs", 197500));
    port.EnregistrerArrivee(new Navire("IM08715871", "MSC PILAR"));
    port.EnregistrerDepart("IM08715871");
    Console.WriteLine("Départ du navire IM08715871 enregistré");
    port.EnregistrerDepart("IM01111111");
    Console.WriteLine("Départ du navire IM01111111 enregistré");
    Console.WriteLine("fin des enregistrements des départs");
}
```

dans la méthode Main :

```
TesterEnregistrerDepart();
```

A l'exécution, vous devriez obtenir ce résultat :

```
Départ du navire IM08715871 enregistré
impossible d'enregistrer le départ du navire IM01111111, il n'est pas dans le port
```



Travail à faire

Que s'est-il passé ?

Posez un point d'arrêt ici : `port.EnregistrerDepart("IM01111111");` et faites une exécution pas à pas détaillé pour voir où s'est déclenchée l'exception et où elle a été attrapée.

✓ Méthode EstPresent(String imo) de la Port

Cette méthode retourne vrai si le navire dont le numéro imo passé en paramètre est dans le port ou non. Vous vous servirez de ce que vous avez écrit plus haut !!!

Pour tester, écrire cette méthode dans la classe Programm

```
public static void TesterEstPresent()
{
    Port port = new Port("Toulon");
    port.EnregistrerArrivee(new Navire("IM09427639", "Copper Spirit", "Hydrocarbures", 156827));
    port.EnregistrerArrivee(new Navire("IM09839272", "MSC Isabella", "Porte-conteneurs", 197500));
    port.EnregistrerArrivee(new Navire("IM08715871", "MSC PILAR"));
    String imo = "IM09427639";
    Console.WriteLine("Le navire " + imo + "est présent dans le port : " + port.EstPresent(imo));
    imo = "IM01111111";
    Console.WriteLine("Le navire " + imo + "est présent dans le port : " + port.EstPresent(imo));
}
```

Et pour exécuter la méthode :

```
TesterEstPresent();
```

Et vous devriez obtenir ceci :

```
Le navire IM09427639est présent dans le port : True  
Le navire IM01111111est présent dans le port : False
```

Fin de la partie 1

