Introduction aux différents outils de debugging de Visual Studio

Mise en place

1. Créer un nouveau projet `TP1Debug` de type application console (.NET FrameWork).
2. Définir le projet `TP1Debug` comme Projet de Démarrage de la solution

Exercice 1 – Création de la classe Pays

Créez la classe `Pays(Id,Nom)` l'id est un entier supérieur ou égal à 1 et le nom doit être non vide et non nul.

En mode sans débogage (F5) déclenchez les deux types d'exception (Id et Nom) et vérifiez l'affichage obtenu. En mode débogage (CTRL+F5) déclenchez les deux types d'exception (Id et Nom) et vérifiez que le débogueur vous renvoie vers la bonne ligne de code à l'origine du problème.

Le mode débogage est donc à privilégier !!

Créez une liste de 3 pays dans le main afin de préparer le suite du TP :

```
Pays p1 = new Pays(1, "France");
Pays p2 = new Pays(2, "Irlande");
Pays p3 = new Pays(3, "Allemagne");
```

Exercice 2 - Erreur / Warning

Mise en place : Ajouter le fichier `TdDebug.cs` au projet.

La première étape du debug est de regarder les erreurs / avertissements indiqués par visual studio.

Supprimez ces erreurs / avertissements dans le code. (On pourra complètement commenter des fonctions).
Corriger les erreurs est indispensable pour compiler. Corriger les warnings peut permettre d'éviter des potentiels erreurs d'exécution, et rend le code plus propre (Suppression du code inutile, etc.)
Note : On peut ajouter "artificiellement" des avertissements dans son code grâce à l'instruction `#warning`. Cela peut être intéressant pour indiquer des portions de code à tester particulièrement, à terminer, etc. Ne cependant pas en abuser, si le projet contient trop de warnings, on ne les regarde plus.

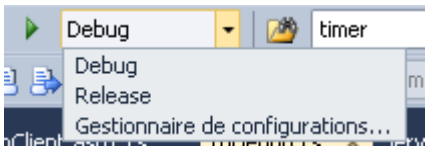
Exercice 3 - Assert

L'assertion permet de vérifier des prédicats. Un cas classique est de vérifier que les pré-conditions des fonctions sont respectées. Les assertions sont très utilisées lors des Tests Unitaires.

.Net propose des fonctions d'assertion qui sont exécutées uniquement en Debug.

Instancier un objet `TdDebug` dans le `Main`. Appeler la méthode `Exercice3()` depuis le `Main`. Compiler en Debug et lancer le programme. Que se passe-t-il ? (Une assertion a échoué).

Compiler en mode "Release" :



Relancer le programme en mode « sans débogage ». Que se passe-t-il ? (Rien, en mode release, version qui sera utilisée pour la production, l'assertion sera ignorée).
Bien revenir en mode Debug.

Note : La méthode `System.Diagnostics.Debug.Assert`, n'est pas à utiliser dans tous les cas. On peut utiliser cette méthode pour par exemple revérifier une saisie utilisateur, mais en aucun cas elle ne doit être la seule validation des données, car ces vérifications ne seront plus faites en Release.

Exercice 4 - Step By Step / Breakpoint Simple / Call Stack

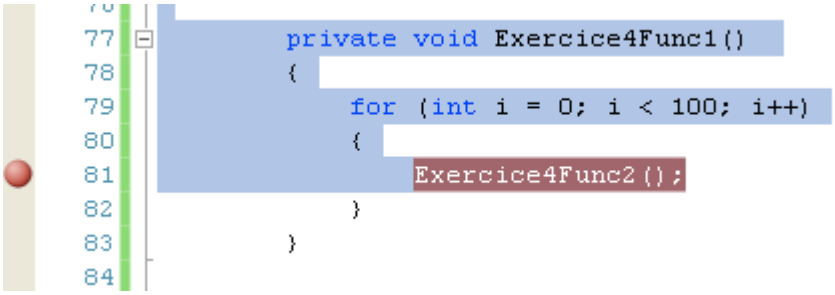
Le debugging étape par étape est un outil que l'on utilise très souvent pour debugger. Pour être efficace avec, il est nécessaire de maîtriser les différentes opérations :

- Pas à Pas Détaillé : Rentrer dans la fonction
- Pas à Pas Principal : Aller à la ligne suivante (ne pas rentrer dans la fonction)
- Pas à Pas Sortant : Aller directement à la fin de la fonction en cours

Il est également nécessaire de savoir ajouter des BreakPoint (points d'arrêt) et naviguer d'un BreakPoint à l'autre.

Exercice :

1. Supprimer l'appel à `Exercice3();`
2. Ajouter un appel à `Exerice4();`
3. Lancer le programme, le programme s'arrête sur un BreakPoint "artificiel"
4. Grace au pas à pas détaillé, aller jusqu'à rentrer dans la fonction `Exercice4Func3()`
5. Utiliser le pas à pas sortant pour sortir rapidement jusqu'à `Exercice4Func1()` (dans la boucle)



6. Ajouter un point d'arrêt dans la boucle et un dans la fonction `Exercice4Func3` :
7. Utiliser le bouton "Continuer" (Fleche verte) pour naviguer entre les 2 BreakPoints.
8. Après quelques tours, supprimer tous les BreakPoint (Menu Deboguer ⇒ Supprimer tous les points d'arrêts);
9. En utilisant le Pas à Pas sortant revenir à la fonction principale (`Exercice4()`).
10. Aller jusqu'à la fin de la fonction en utilisant le Pas à Pas Principal.

Exercice 5 - Watch / Breakpoint Conditionnels

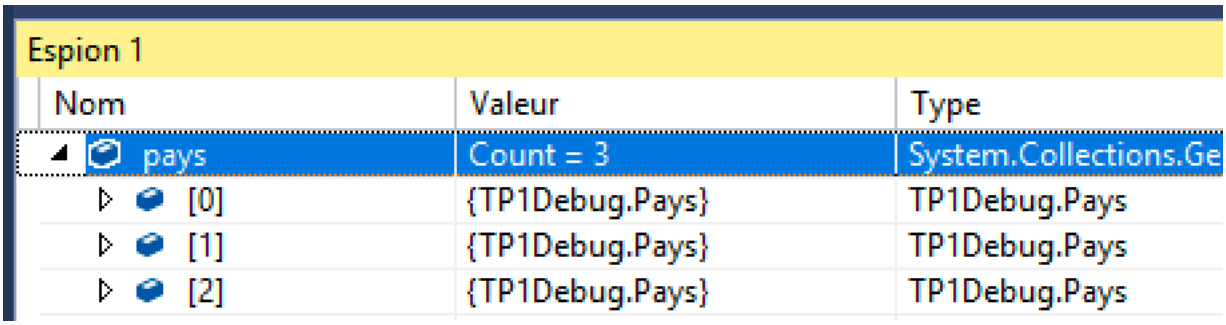
Les espions (ou Watch) permettent d'afficher simplement des variables. Ils sont surtout utiles dans des boucles, pour afficher différentes variables.

Exercice 5.1 - Watch :

- Remplacer `Exercice4()` par `Exercice5_1()`.
- Lancer le programme.
- Ajoute un point d'arrêt à l'instruction `string nom = p.Nom;`
- Continuer l'exécution
- Quand on arrive sur le point d'arrêt, ajouter un espion sur la variable "nom" (Sélectionner la variable ⇒ Clic Droit ⇒ Ajouter un espion)
- Continuer l'exécution
- ⇒ On voit la variable "nom" évoluer

Exercice 5.2 - intérêt de la méthode `ToString()` pour le debug :

- Remplacer `Exercice5_1()` par `Exercice5_2()`.
- Lancer le programme
- Lors de l'arrêt, regarder la variable pays. (Soit avec un espion, soit simplement en survolant la variable avec la souris).



- L'affichage n'est pas très pratique :
- Rajouter une méthode `ToString()` dans la classe Pays qui renvoie le nom et l'id. (ne pas oublier le mot clef `override`)
- Relancer le programme.
- Regarder une nouvelle fois la variable pays.

Exercice 5.3 - BreakPoint conditionnel

- Remplacer `Exercice5_2()` par `Exercice5_3()`.
- Lancer l'exécution
- Sans modifier le code, ni utiliser l'avancement pas à pas, retrouver directement à l'aide d'un BreakPoint la valeur de Fibonacci de 12

Indice : utiliser un BreakPoint conditionnel sur la ligne "`return result;`" de la fonction Fibonacci (clic droit sur le point d'arrêt → Conditions)

Réponse : 144 !

Exercice 5.4 - BreakPoint conditionnel

- Remplacer `Exercice5_3()` par `Exercice5_4()`.
- Lancer l'exécution
- Sans modifier le code, ni utiliser l'avancement pas à pas, retrouver directement le nom du 2ème pays de la liste triée.

Indice : utiliser un BreakPoint par nombre d'accès (Conditions → nombre d'accès) sur l'instruction après "`string nom = p.Nom;`" (le `}`)

Réponse : C'est la France

Exercice 6 - #if DEBUG

En .Net il existe une directive permettant de changer le code que l'on compile selon que le code soit compilé en Debug ou en Release (sans débogage).
Exercice :

- Remplacer l'appel à `Exercice5_4` par l'appel à `Exercice6`
- Lancer le programme en Release. Qu'affiche la console ?
- Lancer le programme en Debug. Qu'affiche la console ?

Note :

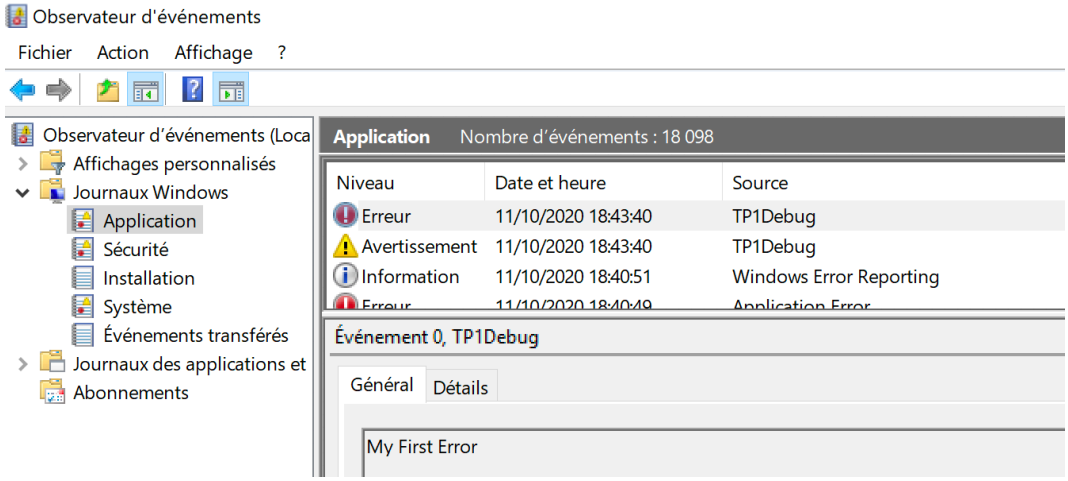
- Le code qui ne sera pas compilé est affiché en grisé dans Visual Studio
- Utiliser cette directive avec parcimonie (afficher plus de détails pour le debug, changer une config, etc.), mais ne pas changer le comportement du programme avec. Le code deviendrait beaucoup plus complexe et difficile à debugger.

Exercice 7 - EventLogger

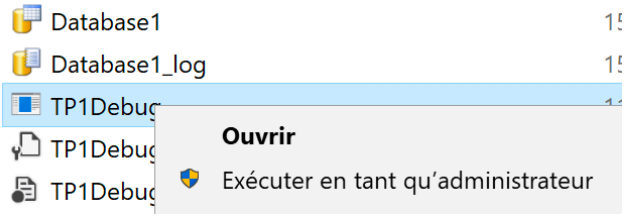
Stocker les events d'un programme est indispensable pour pouvoir trouver un problème lorsqu'une application est en production. Il est important donc de logger toutes les erreurs qui se produisent. (En général, lorsque qu'on catch une exécution, on enregistre un log d'erreur ou d'avertissement). On peut enregistrer ces événements dans une base de données, ou, en .Net directement dans l'Observateur d'Evènements.

Exercice :

- Remplacer l'appel à `Exercice6` par l'appel à `Exercice7`
- Lancer le programme sans débogage (le plus simple est de lancer le .exe généré dans le dossier /bin).
- Afficher l'Observateur d'Evènements (via la commande "***eventvwr***")
- Retrouver les évents que l'on a créés :



Remarque :
Il est possible que cela ne marche pas car il faut les droits d'administrateurs. Dans ce cas là lancez le programme en mode administrateur si Windows vous le permet :



Exercice 8 - Mise en pratique

On va mettre en pratique ces différentes notions.
Définissez un nouveau Projet Console.
Créez une classe `Ingredient(Id,Libelle,quantite)` :

Id >0, Nom en majuscule et non vide, Quantite >0

Ajoutez une base de données LocalDB dans votre projet.
Exécutez le script de création de table dans l'explorateur de serveurs. Actualisez les données.
Exécutez le script d'insertion des données (nouvelle requête sur la table).

Exercice 8.1

Dans votre programme principal créez une liste de courses à partir de la lecture de la base de données.
Utilisez le code suivant (prévoir les using nécessaires et corrigez les erreurs éventuelles à l'aide du debogger) :

```
SqlConnection conn = new SqlConnection("Data Source = (LocalDB)\\MSSQLLocalDB; AttachDbFilename
=|DataDirectory|\\Database1.mdf; Integrated Security = True");

string sql = "SELECT LIBELLE,QUANTITE FROM LISTECOURSES";
//Create a SqlCommand object.
SqlCommand requeteSQL = new SqlCommand(sql, conn);

//Create a DataTable object and open connexion
DataTable dataTable = new DataTable();
conn.Open();

//Run the command by using SqlDataReader.
SqlDataReader rdr = requeteSQL.ExecuteReader();

//Load the data from SqlDataReader into the data table.
dataTable.Load(rdr);

//Display the data from the datatable
foreach (DataRow row in dataTable.Rows)
{
      Console.WriteLine("ingrédient : {0}, quantité {1}", row["LIBELLE"], row["QUANTITE"]);
      // à compléter
}

//Close the SqlDataReader and connexion
rdr.Close();
conn.Close();
```

Affichez cette liste sans l'Id car il devra s'afficher en plus lorsqu'on lance le Programme en Debug.

Exercice 8.2

Ajouter une gestion des erreurs. Lors des échecs d'accès aux bases de données et lors de tous les catchs de votre application, enregistrer l'évènement dans l'Event Viewer de Windows (il se peut que cela ne fonctionne pas …)

Exercice 8.3

Créez une méthode d'insertion d'un nouvel ingrédient `AddIngredient` dans la base de données. Cette méthode renvoie `true` si l'insertion s'est bien déroulée. Rajoutez des assertions sur cette méthode. Bonus : utiliser une procédure stockée