Exercices

Activité_partie 2

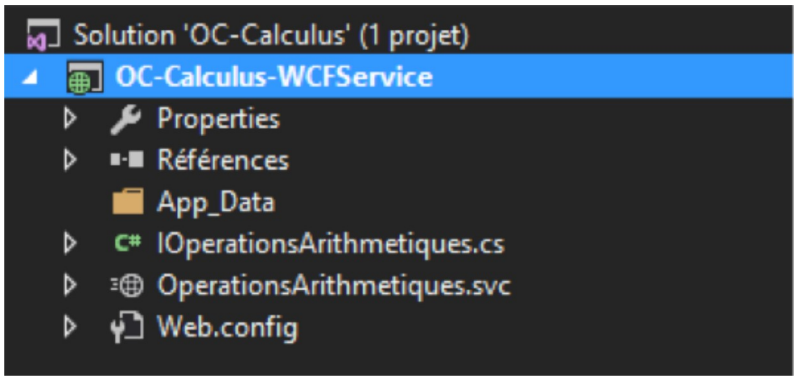
Temps d'attente moyen pour être corrigé sur cet exercice : 33 jours

Retour au cours (https://openclassrooms.com/courses/creez-votre-premiere-application-connectee-en-c-net)

Vous savez maintenant comment déployer et tester un service WCF. Je vous propose donc d'en implémenter un vous-même, qui proposera de réaliser pour une application cliente des calculs mathématiques.

Je vous enjoins donc à télécharger la solution Visual Studio contenue dans cette archive zip (https://static.oc-static.com/prod/courses/files/creez-votre-premiere-application-connectee-en-c-net/OC-Calculus-Sujet.zip) et de la décompresser dans l'endroit de votre choix.

Ouvrez la solution décompressée dans Visual Studio. Vous constaterez que cette dernière ne comprend qu'un seul projet de type WCF Service Application :



Vous remarquerez également si vous essayez de lancer le projet que ce dernier ne compile pas.

En effet, le code que je vous ai fourni est loin d'être complet. Vous devez donc dans cet exercice compléter les méthodes de la classe `OperationsArithmetiques` contenant un commentaire `TODO` en lieu et place de leur implémentation :

```
public double Multiplier(double nb1, double nb2)
{
    //TODO
}

public bool EstMultipleDe(int nb1, int nb2)
{
    //TODO
}

public int ChaineDAdditionsEntieres(string chaine)
{
```

```
    //TODO
}
```

Attention cependant, la tâche semble simple au départ, mais les méthodes vont en se complexifiant. Ainsi les méthodes devant être implémentées sont :

1. `Plus`
   - Renvoie la somme des deux paramètres de type double passés en entrée.
   - Ex : `Plus(42.514d, 13.967d)` doit renvoyer `56.481`
2. `Moins`
   - Renvoie la différence des deux paramètres de type double passés en entrée.
   - Ex : `Moins(42.514d, 13.967d)` doit renvoyer `28.547`
3. `Diviser`
   - Renvoie le résultat de la division du premier paramètre de type double par le second paramètre de type double.
   - Ex : `Diviser(51.75d, 11.5d)` doit renvoyer `4.5`
   - Précision : le cas de la division par 0 n'est pas à traiter.
4. `Multiplier`
   - Renvoie le résultat de la multiplication du premier paramètre de type double par le second paramètre de type double.
   - Ex : `Multiplier(4.5d, 11.5d)` doit renvoyer `51.75`
5. `EstMultipleDe`
   - Renvoie vrai si le premier paramètre de type entier positif est un multiple du second paramètre de type entier positif, sinon renvoie faux.
   - Ex : `EstMultipleDe(45, 5)` doit renvoyer `true`
   - Ex : `EstMultipleDe(60, 13)` doit renvoyer `false`
6. `ChaineDAdditionsEntieres`
   - Renvoie le résultat du calcul passé en paramètre sous forme de chaine de caractères ne pouvant contenir que des nombres entiers positifs ainsi que des signes + (non consécutifs) et ne commençant pas par +. La chaine peut contenir un nombre quelconque d'additions (dont le résultat ne dépassera pas la taille maximale d'un entier).
   - Ex : `ChaineDAdditionsEntieres("68+14+7+29")` doit renvoyer `118`
   - Ex : `ChaineDAdditionsEntieres("48")` doit renvoyer `48`
   - Ex : `ChaineDAdditionsEntieres("")` doit renvoyer `0`
7. `ChaineDAdditionsReelles`
   - Renvoie le résultat du calcul passé en paramètre sous forme de chaine de caractères ne pouvant contenir que des nombres de type double positifs ainsi que le signe +. La chaine peut contenir un nombre quelconque d'additions (dont le résultat ne dépassera pas la taille maximale d'un double) et ne commençant pas par +. Attention ici à bien utiliser le séparateur de décimale approprié à votre localisation (la virgule pour le Français, le point pour l'Anglais, etc.).
   - Ex : `ChaineDAdditionsReelles("68,48+14,21+7,8+2,63")` doit renvoyer `93,12`

- Ex : `ChaineDAdditionsReelles("48,87")` doit renvoyer `48,87`
- Ex : `ChaineDAdditionsReelles("")` doit renvoyer `0`

8. `ChaineDAdditionsEtDeSoustractionsEntieres`
   - Renvoie le résultat du calcul passé en paramètre sous forme de chaine de caractères ne pouvant contenir que des nombres entiers positifs ou négatifs ainsi que les signes + et – (non consécutifs) et ne commençant pas par +. La chaine peut contenir un nombre quelconque d'additions et soustractions (dont le résultat ne dépassera pas la taille maximale d'un entier).
   - Ex : `ChaineDAdditionsEtDeSoustractionsEntieres("-18+65-14+78")` doit renvoyer `111`
   - Ex : `ChaineDAdditionsEtDeSoustractionsEntieres("208+66+14")` doit renvoyer `288`
   - Ex : `ChaineDAdditionsEtDeSoustractionsEntieres("")` doit renvoyer `0`
9. `ChaineDAdditionsEtDeSoustractionsReelles`
   - Renvoie le résultat du calcul passé en paramètre sous forme de chaine de caractères ne pouvant contenir que des nombres réels positifs ou négatifs ainsi que les signes + et – (non consécutifs). La chaine peut contenir un nombre quelconque d'additions et soustractions (dont le résultat ne dépassera pas la taille maximale d'un double) et ne commençant pas par +. Attention ici à bien utiliser le séparateur de décimale approprié à votre localisation (la virgule pour le Français, le point pour l'Anglais, etc.).
   - Ex : `ChaineDAdditionsEtDeSoustractionsReelles("-18,48+65,44-14,484+78,221")` doit renvoyer `110,697`
   - Ex : `ChaineDAdditionsEtDeSoustractionsReelles("208,48315184")` doit renvoyer `208,48315184`
   - Ex : `ChaineDAdditionsEtDeSoustractionsReelles("")` doit renvoyer `0`
10. `ChaineDAdditionsEtSoustractionsEtMultiplicationsEntieres`
    - Renvoie le résultat du calcul passé en paramètre sous forme de chaine de caractères ne pouvant contenir que des nombres entiers positifs ou négatifs ainsi que les signes +, –, * (non consécutifs à l'exception de *- pour représenter la multiplication par un nombre négatif) et ne commençant pas par + ni *. La chaine peut contenir un nombre quelconque d'additions, de soustractions et de multiplications (dont le résultat ne dépassera pas la taille maximale d'un entier). Attention à gérer correctement les priorités !
    - Ex : `ChaineDAdditionsEtSoustractionsEtMultiplicationsEntieres("-69*18+14-22*-75")` doit renvoyer `422`
    - Ex : `ChaineDAdditionsEtSoustractionsEtMultiplicationsEntieres("208*66*14-48*52")` doit renvoyer `189696`
    - Ex : `ChaineDAdditionsEtSoustractionsEtMultiplicationsEntieres("")` doit renvoyer `0`

Une fois les implémentations de ces méthodes réalisées, il vous est fortement recommandé de tester extensivement ces dernières, en effet, elles seront en parties automatiquement corrigées. C'est pour cette même raison que vous ne devez en aucun cas renommer la classe, les fichiers ainsi que modifier les noms et signatures des méthodes précédentes.

Lorsque vous pensez en avoir terminé, placez dans une archive zip les fichiers OperationsArithmetiques.svc et OperationsArithmetiques.svc.cs, si vous avez réalisé des classes supplémentaires, incluez-les. Inutile d'inclure le reste de la solution existante déjà fournie et que vous n'êtes pas censés avoir modifiée.

Une fois votre archive zip faite, envoyez là à l'aide de votre formulaire ci-dessous.

Vous serez évalués sur :

1. La compilation de votre code
   - Absence d'erreur

- Absence d’avertissement

2. La simplicité et la clarté de votre code :
   - Indentation
   - Méthodes bien nommées
   - Variables explicites
   - Concision du code
3. Le respect des consignes :
   - Un client automatique testera votre service web

N’hésitez pas à laisser des commentaires dans votre code pour aider le correcteur dans son travail.

Si vous avez des doutes sur comment compresser vos fichiers (c'est-à-dire, créer un dossier .zip), visionnez les vidéos indiquées dans la partie « Sélection du travail ».

Allez, c’est parti!

Sélection du travail

Votre travail (format **.zip**, 70 Mo max)

Parcourir... Aucun fichier sélectionné.

Comment créer un fichier ZIP sous Windows (https://vimeo.com/album/2826327/video/91856555) ? Sous Mac OS X (https://vimeo.com/album/2826327/video/91856554) ?

Vous pouvez laisser un mot à l'attention de vos correcteurs si vous le souhaitez :

(facultatif)

Validation

Attention Relisez-vous bien ! Après soumission de votre travail, vous ne pourrez plus le modifier ni en renvoyer un nouveau !

Envoyer