420KBB – TP03

Table des matières

420KBB – TP03

Les travaux préparatoires au projet `WAL-1D` arrivent à terme. Votre équipe et vous oeuvrez aux derniers préparatifs et explorez des stratégies pour améliorer la fluidité de certaines opérations. Vous trépignez d'impatience à l'idée de sauver (enfin!) le monde de la pollution endémique contre laquelle vous luttez depuis si longtemps… et vous êtes presque prêt(e)s!

L'attention de votre équipe se portera sur quelques aspects bien spécifiques du projet existant, aussi connu sous le nom de code `TP02`[1].

Aspect A – ajustements à la classe `CanalComm`

Vos collègues ont remarqué que la classe `CanalComm`, qui ne permet en ce moment que de transmettre des `PipelineInfo`, pourrait en fait servir à transmettre n'importe quoi.

Votre tâche : convertir `CanalComm` en classe générique (en gros, passer d'une `CanalComm` à une classe `CanalComm<T>`) et ajuster l'ensemble du code pour tenir compte de ce changement.

Aspect B – ajout d'une classe statique `Chiffrier`

La « comptabilisation » des déchets est présentement faite à même le programme principal, ce qui est contre-productif et complique l'évolution du code (en fait, le programme principal en fait beaucoup trop).

Après en avoir discuté avec vos collègues, vous convenez qu'il faudrait placer cette responsabilité sous l'égide d'une classe à part entière.

Votre tâche : ajouter une classe statique nommée `Chiffrier`. Cette classe offrira les services suivants :

- Une méthode `ComptabiliserCollecte` acceptant en paramètre un symbole (un `char`) et comptabilisant le fait qu'il y a désormais un déchet de plus de cette sorte qui a été collecté
- Une méthode `ObtenirStatistiques` qui aura pour rôle de produire une `string` faite de chaque symbole de déchet collecté suivi du nombre d'occurrences de cette collecte. Par exemple, s'il y a eu collecte de trois `'%'` et de huit `'$'` alors la `string` retournée sera `"% x 3; $ x 8; "` (c'est le format préexistant et vous devez le respecter).

Contrainte : par souci d'efficacité, la méthode `ObtenirStatistiques` doit utiliser un `StringBuilder` pour construire la `string` à retourner.

Contrainte : cette classe doit être sécuritaire à utiliser dans un contexte d'accès concurrent.

Recommandation : implantez cette classe autour d'une propriété privée de type `Dictionary<char,int>` pour vous simplifier l'existence.

[1] Autrement dit : vous faites ces ajustements sur le code du `TP02` (faites une copie de sauvegarde juste au cas!), en profitant du moment pour régler tout bogue résiduel!

Aspect C – ajustements à la classe `Afficheur`

Soit la classe `Afficheur` telle qu'elle était à la fin du `TP02` (note : la vôtre devrait ressembler à ceci mais il se peut qu'elle ne soit pas identique, alors adaptez vos efforts en conséquence). Portez attention aux instructions en caractères gras :

```
class Afficheur
{
   PipelineAffichage Pipeline { get; }
   Surface Surface { get; }
   int terminé = 0;
   Thread FilAfficheur { get; }
   CanalComm<PipelineInfo> CanalComm { get; }
   Messagerie Mess { get; init; }
   Messagerie Info { get; init; }
   Afficheur(Surface surface)
   {
      Surface = surface;
      Pipeline = new PipelineAffichage();
      Pipeline.Ajouter(Pipeline.Appliquer(surface.Cadre));
   }
   public IProjetable Appliquer(Mutable p) =>
      Pipeline.Appliquer(new(), p);

   public Afficheur(Surface surface, CanalComm<PipelineInfo> canalComm,
      Messagerie mess, Messagerie info) : this(surface)
   {
      CanalComm = canalComm;
      Mess = mess;
      Info = info;
      FilAfficheur = new(Exécuter);
   }
   public void Démarrer()
   {
      Interlocked.Exchange(ref terminé, 0);
      FilAfficheur.Start();
   }
   public void Arrêter()
   {
      Interlocked.Exchange(ref terminé, 1);
      FilAfficheur.Join();
   }
   public void Exécuter()
   {
      while (teminé == 0)
      {
         var roboInfo = CanalComm.Balayer();
         if (roboInfo.Count > 0)
```

```
            {
               Mutable surface = Surface.Dupliquer();
               foreach (var info in roboInfo)
                  surface = GénérerHalo(info.Zone, info.Couleur, surface);
               Appliquer(surface);
               Mess.Afficher();
               Info.Afficher();
            }
            Thread.Sleep(500);
         }
         {
            var robotInfo = CanalComm.Balayer();
            if (robotInfo.Count > 0)
            {
               Mutable surface = Surface.Dupliquer();
               foreach (var info in robotInfo)
                  surface = GénérerHalo(info.Zone, info.Couleur, surface);
               Appliquer(surface);
               Mess.Afficher();
               Info.Afficher();
            }
         }
      }
   }
```

Votre équipe et vous-même êtes d'avis que ce code se prêterait mieux à des fonctions asynchrones.

Votre tâche :

- Remplacer tout ce qui a trait à des `Thread` par l'équivalent dans le domaine des `Task`
- Remplacer l'attribut d'instance `terminé` et le code associé par un `CancellationToken` qui sera passé en paramètre à `Exécuter`
- Ajuster l'ensemble de la classe en conséquence

Aspect D – ajustements au programme principal

Enfin, le programme principal doit être adapté pour tenir compte de la nouvelle approche choisie par votre équipe.

Votre tâche s'exprimera en quelques étapes.

Étape 0 : la fonction `CréerFils` du `TP02` doit être remplacée par une fonction `CréerTâches`, de manière à ce que l'appel suivant :

```
Thread[] threads = CréerFils(robots, surf, canal, messagerie, information, src.Token);
```

… soit remplacé par :

```
Task tachesRobot = CréerTaches(robots, surf, canal, messagerie, information, src.Token);
```

`CréerTâches` lancera une fonction asynchrone par robot. Le `Task` retourné par `CréerTâches` sera le résultat d'un appel à `Task.WhenAll` sur l'ensemble des tâches lancées par la fonction, ce qui permettra au code client de savoir quand tous les robots auront conclu leur exécution.

Faites en sorte que le cœur de la fonction asynchrone traitant chacun des robots soit :

```
// ...
while (!décédé && !jeton.IsCancellationRequested && !complété)
{
   (complété, décédé) = await TrouverEtRamasserAsync
   (
      robot, canal, surf, portNo, messagerie, information, jeton
   );
   robot.RéinitialiserPuissance();
}
// ...
```

… où `TrouverEtRamasserAsync` fera une itération de la boucle de traitement du robot.

Étape 1 : bien entendu, vous devrez rédiger aussi la fonction `TrouverEtRamasserAsync`. On parle pour l'essentiel d'une réorganisation du code existant.

Rappel : une lambda peut être asynchrone si on la précède du mot clé `async`, par exemple : `async () => { … }`.

Rappel : vous pouvez transformer une fonction ou une lambda synchrone en son équivalent asynchrone avec `Task.Run`.

Étape 2 : dans le programme principal, convertir `LireTouche` en fonction asynchrone et adapter le code en conséquence.

Étape 3 : faire en sorte que le programme principal s'arrête proprement dès que (a) une touche est pressée ou encore (b) chaque robot a complété sa tâche, que ce soit parce que ses ordures sont ramassées ou parce qu'il a explosé.

Forme du travail

Votre travail prendra la forme d'un seul programme, fait de plusieurs fichiers sources. Il est construit sur les fondations des `TP02`, mais comprend quelques modifications.

Ce programme devra utiliser la bibliothèque de classes développée au `TP00` pour générer des identifiants. Il devra aussi utiliser une bibliothèque de classes fournie par vos chics profs pour gérer une partie de l'affichage.

Vous avez beaucoup de latitude cette fois, mais vous avez aussi peu de temps alors soyez organisé(e) et gardez une copie de sauvegarde du `TP02` au cas où vous devriez revenir en arrière pour trouver ce qui cloche avec les changements apportés dans le `TP03`.

Modalités de remise

Organisation :	Travail individuel ou en équipe de deux
Date de remise :	**Dimanche le 7 décembre 2025 à 23 h 59**
Code source imprimé?	Oui, pour les groupes de Patrice, **mais seulement les parties modifiées pour ce TP**. **Prenez soin de les mettre en relief**, par exemple à l'aide d'un surligneur.
Remise par Colnet?	Oui, dans une archive zip nommée[2] comme suit : `Groupe-NomPrénom-TP02.zip` (p. ex. : `05-TromblonGaetan-TP02.zip`) Cette archive doit contenir votre projet **une fois celui-ci nettoyé** (demandez à votre professeur si vous ne comprenez pas cette partie de la consigne)

Amusez-vous bien!

[2] http://h-deb.ca/CLG/Cours/demander-aide.html#remise_travaux