Conception Objet et Programmation *C#*,
IUT département Informatique,
1ère année, 2015-2016

Conception Objet et Programmation en *C#*

Devoir à la Maison n° 1
Delegates et Découplage

Exercice 1 On considère une application interactive permettant de visualiser l'état de quatre automates (exemple : distributeur de boisson, vente de billets *SNCF*, commande de repas de restauration rapide, etc.) en libre service. Les automates peuvent être libres ou occupés. Les automates sont numérotés de 0 à 3. L'affichage de l'état des automates se fait en affichant un `O` pour un automate occupé, et un `L` pour un automate libre. Par exemple, à un instant t, on peut avoir l'état du système suivant :

```
-----------------------------
| Automate | 0 | 1 | 2 | 3 |
| État     | L | O | O | L |
-----------------------------
```

FIGURE 1 : La vue d'affichage en mode console de l'état du système.

a) Concevoir une classe (ou un ensemble de classes) métier pour représenter l'état du système (l'état libre ou occupé de chacun des 4 automates). L'objectif de ces classes est seulement de représenter les données (aussi appelé *modèle*) de l'application, mais pas d'implémenter les fonctionnalités d'*Interface Homme Machine* (en abrégé *IHM*).

b) Nous voulons maintenant implémenter une ou plusieurs classes qui permettent d'implémenter une *IHM* pour cette application. Cette classe permet les fonctionnalités suivantes :

- Affichage du modèle avec une vue comme représenter sur la figure 1
- Saisie par l'utilisateur du nouvel état d'un automate (indice de l'automate concerné et nouvel état).

c) Implémenter un programme permettant, en s'appuyant sur les classes développées dans les questions précédentes, d'itérer (tant que l'utilisateur le souhaite) les opérations suivantes :

1. Saisie du nouvel état d'un automate ;
2. Mise à jour du modèle de données représentant l'état du système ;
3. Mise à jour de la vue pour afficher le nouvel état du système.

d) Dans le but de faire une simulation du traitement des demandes du public, afin de s'assurer que le temps d'attente aux guichets et face aux automates reste raisonnable ou s'il faut envisager d'ajouter un automate, on souhaite changer le fonctionnement de l'application comme suit :

- À chaque itération, un client arrive pour utiliser un automate. Il est inséré dans une file d'attente (collection initialement vide). On attends 1 seconde (utiliser par exemple `System.Threading.Thread.sleep()`. Voir exemple de code ci-dessous) avant l'itération suivante.

- Chaque automate est occupé pendant un temps aléatoire (durée en millisecondes aléatoire entre 0 et 8 secondes) à partir de l'instant où il commence à traiter un client. Dès qu'il a fini, il provoque une mise à jour de la vue (appel immédiat de la méthode d'affichage dans la console de la question b)). On pourra utiliser les fonctionnalités du package `System.Threading.Tasks` (Voir exemple de code ci-dessous). Il commence aussitôt à traiter le client suivant dans la file d'attente, en le supprimant de la file et en tirant une nouvelle durée aléatoire. Si aucun client n'est en attente, il boit une gorgée de café en attendant l'itération suivante avec l'arrivée d'un nouveau client (dans moins d'une seconde).

Représenter un diagramme de classes de l'application développées dans les questions précédentes. On supposera que les *package* `Metier` contient les classes de la question a), permettant de représenter l'état du système (modèle), et qu'un *package* `ApplicationConsole` contient les classes des questions b) et c) (*IHM* et application). Quel problème constatez-vous ? On rappelle qu'il ne peut pas y avoir de dépendance cyclique entre les *packages*, ce qui se traduit, si on essaye de créer des projets *Metier* et *ApplicationConsole* distincts dans *Visual Studio*, par une erreur de dépendance des modules.

e) On se propose de modifier la classe qui gère le modèle (question a)) pour qu'elle contienne une instance d'un type *delegate* (en d'autre terme la référence vers une méthode) dont l'invocation déclenche la mise là jour de la vue et le traitement du client suivant de la file. Ainsi, à l'issue de son temps d'attente, le guichet invoquera l'instance de *delegate*, provoquant instantanément le traitement adapté. Modifier le diagramme de classes pour éliminer les dépendances cycliques en utilisant cette idée.

Exemple de planification de tâche. Voici un exemple de code qui planifie l'exécution d'un code après une attente d'un nombre de millisecondes donné. Ici, le code est juste un affichage.

```
1  using System;
2  using System.Threading;
3  using System.Threading.Tasks;
4
5  namespace ExemplePlanification
6  {
7   public class ExemplePlanification
8   {
9
10   // Méthode qui fait un affichage après un certain temps
11   // Nombre de millisecondes passé en paramètre.
12   public static void WaitForAWhile(int nMilliseconds)
```

```
13    {
14      Task.Run( () => {          // Code du thread
15                                 Thread.Sleep(nMilliseconds);
16                                 Console.WriteLine("Hello, I had such a good nap!");
17                               } );
18
19    }
20
21    // Le Main
22    public static void Main(string[] args)
23    {
24      Console.WriteLine("Hello, I go to sleep...");
25      WaitForAWhile(3000);
26      Console.WriteLine("In the meantime...");
27
28      // Attendre la fin de la tâche avant de terminer le programme
29      Thread.Sleep(4000);
30      Console.WriteLine("Well, that's the end folks...");
31    }
32  }
33 }
```

La trace d'exécution de ce programme est :

```
Hello, I go to sleep...
In the meantime...
Hello, I had such a good nap!
Well, that's the end folks...
```