Cas Synapse

# PRÉSENTATION

SynapsInfo est une SSII assez importante spécialisée en informatique de gestion. Ses collaborateurs (plus de 200 salariés en décembre 2006) interviennent sur les sites de ses différents clients dans le cadre de projets.

# LOGICIEL MISSION SOFT

**À utiliser : annexes 1 à 5**.

Afin de gérer au mieux les affectations de ses différents collaborateurs informaticiens ainsi que le coût et la rentabilité des différents projets, SynapsInfo a commencé à développer un logiciel en interne. Après un test en grandeur réelle dans l'entreprise, SynapsInfo désire commercialiser ce logiciel. Son  nom de code provisoire est « MissionSoft ».

Un projet est constitué de plusieurs missions. Chaque projet a un coût de main d’œuvre facturé. Une mission est prise en charge par un intervenant de SynapsInfo. Pour chaque mission, l’intervenant concerné remplit un relevé horaire (exemple en ***annexe 4***) dans lequel il précise pour chaque jour, le nombre d'heures effectuées pour la mission. Les ***annexes 1, 2 et 3*** présentent les classes métiers et techniques utilisées.

L’un des objectifs de l’application est de comparer le coût facturé et le coût réel de la main d’œuvre.

|  |  |
| --- | --- |
|  | **Travail à faire** |
| 1 | Écrire la méthode *margeBruteCourante()* de la classe Projet. |
| 2 | Écrire la méthode *cumulCoûtMO()* de la classe Projet. |
| 3 | Écrire la méthode *nbHeuresEffectuées()* de la classe Mission. |

Le développeur responsable de la classe Mission a écrit des tests unitaires de différentes méthodes. Ces tests ont pour but de vérifier que les méthodes se comportent comme prévu. Pour cela le développeur utilise une classe dédiée aux tests, la classe *Assert*.

Un test unitaire est considéré comme conforme si durant son exécution aucune assertion (affirmation) n’est violée. L’***annexe 5*** décrit plus précisément l’utilisation de la classe Assert.

Le développeur envisage de tester la méthode *ajoutRelevé()* de la classe *Mission*. Cette méthode ajoute une date et un nombre d'heures au relevé ; une vérification est effectuée afin de ne pas enregistrer plus de 8 heures pour une date. Voici le code actuel de cette méthode :

public void ajoutRelevé(DateTime jour , int nbHeures ) // ajoute au relevé, une date et un nombre d’heures

{ if (nbHeures <= 8)

{ releveHoraire.Add(jour, nbHeures); }

else

{ releveHoraire.Add(jour,8) ; } }

Pour tester que la méthode ajoute correctement une date et un nombre d'heures, le développeur a écrit un programme dont voici un extrait significatif :

public void TestMethod1()

{

Mission laMission ;

DateTime laDate1, laDate2 ;

int nbHeures ;

laMission = new Mission("Client","Conception vecteur",15,new Dictionary<DateTime,int>(),new Intervenant("Dupont",45)); // instanciation d’une mission

laDate1 = new DateTime(2007,4,4);

laMission.ajoutRelevé(laDate1,5);

nbHeures = laMission.RelevéHoraire[laDate1];

Assert.AreEqual(5,nbHeures,"résultat non conforme");

laDate2 = new DateTime(2007,4,6);

laMission.ajoutRelevé(laDate2,9);

nbHeures = laMission.RelevéHoraire[laDate2];

Assert.AreEqual(8,nbHeures,"résultat non conforme");

Le résultat des deux tests est satisfaisant (pas de message affiché).

Le développeur poursuit les tests car les spécifications de la méthode *ajoutRelevé* précisent que dans le cas où deux relevés portent sur la même date, les heures doivent être cumulées, sans toutefois dépasser 8 heures. L’extrait du programme est le suivant :

…

DateTime laDate = new DateTime(2007,4,4);

laMission.ajoutRelevé(laDate,6);

laMission.ajoutRelevé(laDate,5); // ajout d’un relevé pour la même date

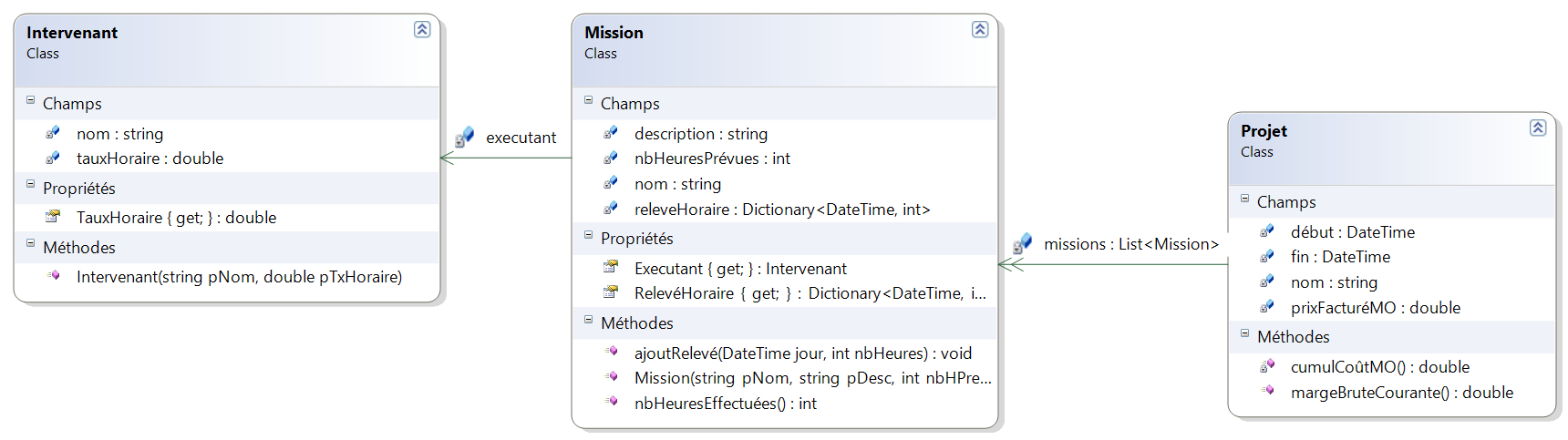
nbHeures = laMission.RelevéHoraire[laDate];

Assert.AreEqual(8,nbHeures,"résultat non conforme");

Le résultat du test, cette fois, n'est pas satisfaisant (affichage du message “résultat non conforme”).

|  |  |
| --- | --- |
|  | **Travail à faire** |
| 4 | 1. Expliquer la cause de l'échec du test unitaire. 2. Réécrire le corps de la méthode *ajoutRelevé()* afin de corriger l'erreur. |
|  | 1. A la suite du test pécédent, tester le résultat de la méthode *nbHeuresEffectuées()* de la Mission laMission. |

#### ANNEXE 1 : Diagramme des classes métiers



#### ANNEXE 2 : Descriptif des classes métiers

***Seules figurent les méthodes utiles à l’étude au traitement du dossier 4.***

public class Intervenant

{

private string nom ;

private double tauxHoraire;

public Intervenant(string pNom, double pTxHoraire) // accesseur

{ }

public double TauxHoraire // accesseur

{ }

}

public class Projet

{

private string nom;

private DateTime début;

private DateTime fin;

private double prixFacturéMO;// prix auquel le projet a été vendu pour la main d’œuvre

private List<Mission> missions;

private double cumulCoûtMO()

// retourne le coût cumulé des heures de main d’œuvre effectuées pour l’ensemble

// des missions du projet

{ }

public double margeBruteCourante()

// retourne la différence entre le prix facturé au chapitre main d’œuvre et le coût des

// heures de main d’œuvre effectuées pour l’ensemble des missions du projet

{ }

}

public class Mission

{

private string nom;

private string description ;

private int nbHeuresPrévues ; // nombre d'heures prévues pour réaliser la mission

private Dictionary<DateTime,int> releveHoraire ; // nombre d'heures passées par jour par la personne chargée d'exécuter cette mission

private Intervenant executant ; // personne chargée d'exécuter la mission

public Dictionary<DateTime, int> RelevéHoraire // accesseur sur l’attribut relevéHoraire

{ }

public Intervenant Executant // accesseur sur l’attribut exécutant

{ }

public Mission(string pNom, string pDesc, int nbHPrevues,Dictionary<DateTime,int> relHor, Intervenant exec)

{ }

**public void ajoutRelevé(DateTime jour , int nbHeures )** // ajoute au relevé, une date et un nombre d’heures

{ }

**public int nbHeuresEffectuées()** // retourne le nombre d’heures réellement effectuées du relevé horaire

{ }

}

#### ANNEXE 3 : Descriptif des classes techniques

## La classe Dictionary <TKey, TValue>

* **Tkey :** Type des clés dans le dictionnaire.
* **Tvalue :** Type des valeurs dans le dictionnaire.

*// collection d’éléments (clé, valeur) permettant d’extraire une valeur (de type TValue) à partir de*

*// sa clé (de type TKey) ; à une clé présente dans le dictionnaire correspond une et une seule valeur*

[**Propriétés**](javascript:void(0))

|  |  |
| --- | --- |
| [Count](http://msdn.microsoft.com/fr-fr/library/vstudio/zhcy256f.aspx) | Obtient le nombre de paires clé/valeur contenues dansDictionary<TKey, TValue>. |
| [Keys](http://msdn.microsoft.com/fr-fr/library/vstudio/yt2fy5zk.aspx) | Obtient une collection contenant les clés dans Dictionary<TKey, TValue>. |
| [Values](http://msdn.microsoft.com/fr-fr/library/vstudio/ekcfxy3x.aspx) | Obtient une collection contenant les valeurs deDictionary<TKey, TValue>. |

[**Méthodes**](javascript:void(0))

|  |  |  |
| --- | --- | --- |
| [Add](http://msdn.microsoft.com/fr-fr/library/vstudio/k7z0zy8k.aspx) | Ajoute la clé et la valeur spécifiées au dictionnaire. | monDic.Add(« Tintin »,3) |
| [ContainsKey](http://msdn.microsoft.com/fr-fr/library/vstudio/kw5aaea4.aspx) | Détermine si Dictionary<TKey, TValue> contient la clé spécifique. | if(monDic.ContainsKey(« Tintin »)) |
| [ContainsValue](http://msdn.microsoft.com/fr-fr/library/vstudio/a63811ah.aspx) | Détermine si Dictionary<TKey, TValue> contient une valeur spécifique. | if(monDic.ContainsValue(3)) |

**Accès à une valeur du dictionnaire grâce à la clé**monDic de type Dictionary<string,int> **:**

string clé = « Tintin » ;

int nb = monDic[clé] ;

monDic[clé] = 3 ;

**Parcours d’un dictionnaire :**

* Récupérer la collection (nommée valueColl) de **valeurs** présentes dans le dictionnaire monDic de type Dictionary<string,int>

Dictionary<string,int>.ValueCollection valuColl = monDic.Values;

* Parcourir la collection grâce à l’instruction foreach

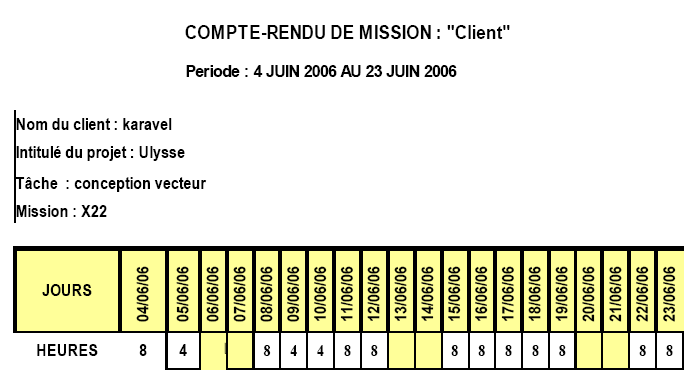
foreach (int nb in valuColl)

{

Console.WriteLine ( « Valeur : {0}», nb);

}

#### ANNEXE 4 : Exemple de relevé horaire



#### ANNEXE 5 : Classe de test Assert du framework .Net

La classe technique *Assert* dispose d’une méthode à portée classe (statique) permettant de vérifier si une affirmation est ou non respectée : si oui, rien ne se passe, sinon, le message d'erreur passé en paramètre est affiché.

public static void Assert.AreEqual (int expected, int actual, String message)

// vérifie si les valeurs entières expected et actual sont égales.   
 // Affiche un message si l'assertion échoue

**Exemple d’utilisation de la classe Assert :**

Assert.AreEqual (72, 8\*9 , “résultat non conforme”) *// n’affiche rien puisque 8\*9 est bien égal à 72*

Assert.AreEqual (83, 9\*9 , “résultat non conforme”) *// affiche le message puisque 9\*9 = 83 est faux*