TP commerciaux

*Créé par Patrice Grand*

# Liens avec le référentiel SIO en SLAM4

Utilisation et enrichissement de bibliothèques de composants logiciels et à la programmation, notamment au sein d’un *framework* (Visual Studio).

**Savoir-faire**

* Programmer un composant logiciel
* Exploiter une bibliothèque de composants
* Adapter un composant logiciel
* Programmer au sein d’un *framework (*ici Visual Studio*)*

**Savoirs associés**

* Persistance et couche d’accès aux données, technologies et techniques associées (ici sérialisation XML)
* Techniques de test unitaire et d’intégration d’un composant logiciel
* A4.1.6 Gestion d’environnements de développement et de test
* A4.1.8 Réalisation des tests nécessaires à la validation d’éléments adaptés ou développés
* A4.2.3 Réalisation des tests nécessaires à la mise en production d’éléments mis à jour

La société **Camaurel** fabrique et vend des placards et rangements à monter soi-même. Son siège est installé dans la région parisienne ; un nombre important de commerciaux rayonne en France afin de développer et consolider sa clientèle : principalement des distributeurs (magasins d'aménagement, surfaces de bricolage). Pendant ses déplacements, le commercial est amené à engager des frais, pour ses transports, ses repas ou les nuits passées à l'hôtel. L'application proposée ici vise à gérer ces notes de frais, documents rendus justifiant des frais engagés.

# Gestion des commerciaux

Chaque commercial a un nom, un prénom, une catégorie professionnelle – pour laquelle un barème pour certaines notes de frais sera appliqué, de type caractère-, une puissance de voiture de type entier – qui permettra d’affecter un tarif pour les frais de transport -.

Par ailleurs chaque commercial est affecté à un service, ici le service commercial.

# Gestion des notes de frais de chaque commercial

Régulièrement après chaque mission, le commercial remet ses notes de frais.

Le système a besoin de connaître, à chaque création d'une note de frais, la date du frais, le montant à rembourser (champ de type double) et si chaque note de frais a été remboursée; chaque note est identifiée par un numéro. On désire également que chaque note de frais *puisse connaître* le commercial qui l’a émise.

Le service commercial ne contient pas de champ particulier.

**Travail à faire**

Faire le diagramme de classes sans faire apparaître les méthodes.

# Code de la classe Commercial

**Travail à faire**

* Ecrire le code de la classe Commercial. Cette classe expose un accesseur sur chacun de ses champs, une méthode d’ajout d’une note de frais – void ajouterNoteFrais(NoteFrais f)- ainsi qu’une redéfinition de la méthode ToString qui permettra d’afficher :

Commercial c;

c = new Commercial("Jean", "Dupond", 25, 'A');

Console.WriteLine(c.ToString());



# Code de la classe NoteFrais

* Cette classe expose des accesseurs sur les champs montantARembourser et leCommercial.
* Un modificateur permet de passer à true le champ rembourse et un second modificateur valorise le champ montantARembourser, tel que :

public void setMontantARembourser()

{

this.montantARembourser = calculMontantARembourser();

}

virtual public double calculMontantARembourser() { return 0; }

Cette dernière méthode sera implémentée plus tard pour les classes dérivées.

Enfin une redéfinition de la méthode ToString permettra d’écrire et d’afficher :



**Remarque** : le numéro de la note de frais est géré automatiquement ; dans notre cas c’est la première note de frais pour ce commercial.

**Travail à faire**

* Ecrire le code des méthodes de la classe NoteFrais.
* Faire un test sur l’ajout des notes de frais à un commercial. Vérifier que le test suivant passe :

|  |  |
| --- | --- |
|  |  |

# Gestion de la classe ServiceCommercial

Cette classe contient – dans un premier temps - 3 méthodes :

* Son constructeur par défaut, à écrire,
* Une méthode qui ajoute un commercial void ajouterCommercial(Commercial c)
* Une méthode int nbFraisNonRemboursés()qui retourne le nombre de frais dans l’état non remboursé pour tous ses commerciaux.

**Travail à faire**

* Ecrire la classe ServiceCommercial, effectuer le test unitaire suivant et vérifier qu’il passe bien :

|  |  |
| --- | --- |
|  |  |

# Les 3 classes dérivées de la classe NoteFrais

Il y a trois types de notes de frais.

* Les **notes de transport** routier dues au carburant (les péages sont pris en charge par une carte de société). Le commercial précise sur sa note le nombre de km, sans autres justifications ; le remboursement se fait sur la base de la puissance de la voiture : 0.1 €/km pour une voiture de moins de 5 chevaux, 0.2 € pour une voiture entre 5 et 10 chevaux et 0.3 € au-delà de 10 chevaux.
* Les **notes de repas** de midi. Le commercial doit fournir la facture du repas. Le remboursement se fait en tenant compte de la catégorie du commercial (échelle interne à la profession) ; la catégorie A donne lieu à un remboursement de 25 €, la catégorie B donne lieu à un remboursement de 22 € et la catégorie C à un remboursement de 20 €. Dans le cas où la facture n'atteint pas ce plafond, le commercial est remboursé du montant de la facture produite.
* Les **notes de nuitée**, comprenant également le repas du soir et le petit déjeuner. Deux éléments interviennent ; la catégorie pour laquelle la même règle est appliqué que pour les repas de midi (A 🡪 65 € , B 🡪 55€, C 🡪 50€ ) et la région touristique dans laquelle se trouve l'hôtel. Un coefficient est appliqué aux tarifs :
  + 0.90 pour la région 1,
  + 1 pour la région 2
  + 1.15 pour la région 3.

Le plafond indiqué (forfait et coefficient) n'est remboursé que si le montant de la facture lui est supérieur.

**Travail à faire**

* Ajouter les nouvelles classes au diagramme de classe
* Ecrire la déclaration des classes et le code des trois constructeurs. Tester dans le Main une création des trois types de frais.

## 6.1 La classe FraisTransport

Cette classe redéfinit la méthode virtuelle calculMontantARembourser. C'est cette méthode -redéfinie- qui calculera les réels montants des remboursements pour les classes dérivées ; la méthode *setMontantARembourser* sera appelée à la construction des instances des classes dérivées.

**Travail à faire**

* Ecrire le code de la méthode calculMontantARembourser , tester avec :

|  |  |
| --- | --- |
|  |  |

* Redéfinir la méthode ToString permettant d’afficher :

static void Main(string[] args)

{

Commercial c;

c = new Commercial("Jean", "Dupond", 8, 'A');

NoteFrais f;

f = new FraisTransport(new DateTime(2013, 11, 12), c,250);

Console.WriteLine(f.ToString());



## 6.2 La classe *RepasMidi*

**Travail à faire**

* Ecrire la méthode calculMontantARembourser, faire deux tests dans l’environnement de tests :
  + En prenant 35 comme montant du repas, pour le même commercial que dans test précédent, vous devez obtenir 25 euros de remboursement.
  + En prenant 15 comme montant du repas, vous devez obtenir 15 euros de remboursement
* Redéfinir la méthode ToString

## 6.3 La classe *Nuitee*

**Travail à faire**

* Ecrire la méthode calculMontantARembourser , faire à nouveaux deux tests pour le commercial de catégorie ‘A’ :
  + Avec un montant de 46 euros dans la région 2 ; vous devez obtenir 46 euros de remboursement
  + Avec un montant payé de 80 euros dans la région 3 ; vous devez obtenir 74,75 euros de remboursement
* Redéfinir la méthode ToString

# Gestion des notes de frais d'un commercial

**Travail à faire**

* Ajouter une méthode dans la classe Commercial qui permet d’obtenir le cumul des notes de frais remboursées pour une année passée en paramètre.

On désire pouvoir ajouter des notes de frais à partir du service commercial. C’est ainsi que l’appel suivant pourra ajouter différentes notes de frais :

ServiceCommercial sc;

Commercial c1 ;

sc = new ServiceCommercial();

c1 = new Commercial("Dupond", "Jean", 7, 'B');

sc.ajouterCommercial(c1);

sc.ajouterNote(c1, new DateTime(2013, 11, 15), 100)); *// ajoute un frais de transport*

sc.ajouterNote(c1, new DateTime(2013, 11, 21), 15.5)); *// ajoute une note de repas*

sc.ajouterNote(c1, new DateTime(2013,11 , 25), 105, 2); *// ajoute une nuitée*

**Travail à faire**

Ecrire les méthodes nécessaires dans la classe ServiceCommercial. Tester en imaginant des méthodes de test.

# Sérialisation

Nous allons créer une classe **PersisteCommercial** qui permettra d'écrire les appels suivants :

PersisteServiceCommercial.sauve(sc, "service.sr"); *// le ServiceCommercial sc est sérialisé et enregistré en mémoire*

ServiceCommercial sc1 = PersisteServiceCommercial.charge("service.sr"); *//le ServiceCommercial sc1 est désérialisé*

**Travail à faire**

En utilisant la sérialisation binaire, écrire la classe PersisteCommercial qui ne contiendra que les deux méthodes statiques illustrées ci-dessus.