BTS SERVICES INFORMATIQUES AUX ORGANISATIONS

E5 : Production et fourniture de services informatiques

**SESSION 2018**

**Durée : 4h00 Coefficient : 5**

Étude de cas

CAS DÉMGUIV

Ce sujet comporte 5 pages dont un dossier documentaire de 3 pages. La candidate ou le candidat est invité.e à vérifier que le sujet qui lui a été remis est complet.

Aucun matériel ni document autorisé

**Dossier documentaire :**

Table des matières

[1. Extrait du diagramme de classes métier 4](#_Toc4442524)

[2. Description textuelle des classes métier 4](#_Toc4442525)

[3. Exemple d'utilisation d'une collection 4](#_Toc4442526)

[4. Exemple d'utilisation de tests unitaires 4](#_Toc4442527)

[5. Evolution et ajout d’une classe Manager 5](#_Toc4442528)

**PRÉSENTATION DU CONTEXTE**

***L'organisation cliente :* DÉMGUIV**

DÉMGUIVest une entreprise familiale spécialisée dans le déménagement de particuliers interurbain et intra-urbain. Fondée par M. GUIVARCH en 1980 dans la ville de Quimper, la société DÉMGUIVn'a cessé de se développer et compte aujourd'hui trois nouvelles agences : Rennes, Nantes et Angers. Le siège social de la société est situé à Quimper dans les mêmes locaux que l’agence. Chaque agence est dirigée par un responsable d'agence.

La société DÉMGUIV propose plusieurs formules de déménagement réservées aux particuliers :

* La formule économique dans laquelle les déménageurs s'occupent uniquement du chargement et de la livraison des meubles et des cartons.
* La formule standard comprend toutes les prestations de la formule économique avec démontage et remontage des meubles si cela s'avère nécessaire. Les cartons sont fournis par l'entreprise.
* La formule de luxe comprend toutes les prestations de la formule standard en plus de la mise en cartons et du déballage.
* La formule sur mesure comprend toutes les prestations de l'une des trois formules précédentes au choix de la cliente ou du client, enrichies par ses besoins spécifiques. Par exemple, la cliente ou le client peut choisir la formule économique en déléguant l'emballage des objets fragiles à l'entreprise DÉMGUIV.

***L'entreprise prestataire de services :* LOGIMOUVE**

LOGIMOUVE est une entreprise de services du numérique (ESN) nantaise spécialisée dans le développement de logiciels destinés aux professionnels du transport. Elle assure, en outre, la formation et l'assistance pour sa gamme de logiciels. Fondée en 2003, elle compte quinze salariés dont dix développeurs.

***Le projet***

Afin d'optimiser la gestion de ses contrats de déménagements, la société DÉMGUIV a fait appel en 2013 à un prestataire informatique pour se doter d'un logiciel de gestion des déménagements. En 2014, le logiciel *PSDEM (Planification-Suivi Déménagement)* a été déployé dans les agences de DÉMGUIV*.*

L'application *PSDEM* est développée en langage *Java*. Elle est composée :

* d’un module de gestion des dossiers de déménagements : *SUIVI\_DEM* ;
* d’un module de planification des déménagements : *PLAN\_DEM* ;
* d’un module de gestion des absences des salariés : *GEST\_ABS*.

L’application *PSDEM* est installée dans les quatre agences et s’appuie sur une base de données située à Quimper accessible depuis toutes les agences.

L'utilisation de *PSDEM*a révélé des dysfonctionnements qui n'ont pu être corrigés suite à la liquidation judiciaire de son éditeur initial. L'entreprise DÉMGUIV a alors confié la correction et l'évolution de l'application à l'ESN LOGIMOUVE.

Nouvellement embauché.e dans l'entreprise LOGIMOUVE, vous êtes chargé.e de participer aux différentes missions du projet d’évolution de l’application *PSDEM*.

**Mission 3 : Gérer les absences des salariés**

***Documents à utiliser : 1,2,3, 4 et 5 + classes Java***

*IMPORTANT : la candidate ou le candidat doit développer* ***en Java. Il est demandé le code complet correspondant à la question + copie écran code correspondant sur Eclipse***

Le module *GEST\_ABS* alimente la base de données en ce qui concerne les absences des salariés. Sa version actuelle est très sommaire, une nouvelle version est à l’étude.

La durée d'une absence est exprimée en nombre de jours ouvrables. Certaines absences (exemple : congés annuels) nécessitent une demande d'autorisation que la ou le salarié.e doit formuler. D'autres absences, comme les arrêts de travail, ne nécessitent pas de demande préalable.

La nouvelle version du module *GEST\_ABS* permettra :

* aux salariés, de saisir leurs demandes d'autorisation d'absence, de visualiser l'état de leurs demandes (en attente, acceptée ou refusée), de visualiser l'historique de leurs absences et de consulter l'état de leurs différents soldes ;
* aux secrétaires, de saisir les arrêts de travail des salariés ;
* aux responsables d'agence, de consulter les demandes d'autorisation d'absences et d'y répondre, de consulter l'état des soldes, de visualiser les absences sous la forme de plannings hebdomadaires et mensuels ;
* aux chefs d'exploitation, de visualiser les absences des déménageurs sous la forme de plannings hebdomadaires et mensuels.

Un planning d'absences fait apparaître de manière claire (un code couleur a été défini pour représenter les motifs d'absence) :

* les absences qui ne nécessitent pas de demande d'autorisation ;
* les absences dont la demande d'autorisation est en attente ;
* les absences dont la demande d'autorisation est acceptée.

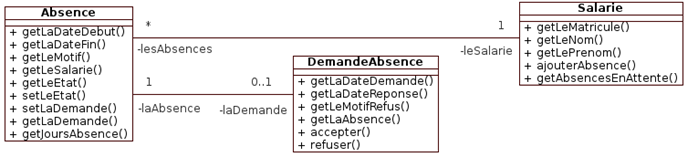
L'itération courante du projet a pour objectif la visualisation du planning mensuel des absences.

**Votre mission**

1. **Implémentez la méthode *ajoutUneAbsence* de la classe** Salarie
2. **Ecrire un test unitaire dans la classe** TestSalarie **qui vérifie que la méthode *ajoutUneAbsence* fonctionne**
3. **Fournir le code de la méthode *refuser* de la classe** DemandeAbsence**.**
4. **Dans la classe de test unitaire** TestDemandeAbsence**, ajoutez une méthode de test de la méthode *refuser* de la classe** DemandeAbsence***. On vérifiera que le motif et l’état de l’absence ont bien été affectés.***
5. **Fournir le code de la méthode** *getAbsencesEnAttente* **de la classe** Salarie**.**
6. **Dans la classe de test unitaire** TestSalarie**, ajoutez une méthode de test de la méthode *getAbsencesEnAttente* de la classe** Salarie
7. **Répondez à l’évolution du besoin décrite dans le document 5**
8. **Ecrire un test unitaire de la méthode *decideConges* de la classe** Manager

**Dossier documentaire**

## Extrait du diagramme de classes métier



## Description textuelle des classes métier

Les classes sont directement fournies sous forme de classe Java, à compléter.

## Exemple d'utilisation d'une collection

**Exemple avec interface List et classe ArrayList**

Salarie salarie = new Salarie("D0001","LEGRAND","Paul") ;

List<Salarie> lesSalaries ; *// Déclaration d'une collection d'instances de la classe Salarie*

lesSalaries = new ArrayList<Salarie>() ; *// Instanciation d'une collection*

lesSalaries.add( salarie ) ; *// Ajout d'un salarié*

System.out.println( lesSalaries.get( 0 ) ) ; *// Affichage du premier élément*

System.out.println( lesSalaries.size() ) ; *// Affichage du nombre d'éléments*

for( Salarie unSalarie : lesSalaries ){ *// Parcours de la collection (foreach dans d’autres langages)*

System.out.println(unSalarie.getNom()) ; *// Affichage de l'élément courant*

}

}

lesSalaries.contains(salarie) : retourne vrai si la collection lesSalaries contient salarie, faux sinon.

## Exemple d'utilisation de tests unitaires

**Exemple avec librairie Junit**

La classe **Assert** contient différentes méthodes statiques permettant de savoir si le test unitaire a réussi ou non. Elle contient la méthode ***assertEquals*** dont voici la signature :

public static void **assertEquals**([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html) message,

long expected,

long actual)

message : message à afficher si l'assertion échoue, c'est-à-dire lorsque la valeur attendue est différente de la valeur obtenue.

expected : valeur de type double contenant la valeur attendue

actual : valeur de type double contenant la valeur obtenue

Exemple : assertEquals(« test KO »,val1,val2) ;

**Exemple avec librairie Hamcrest**

La librairie Hamcrest étend le principe des assertions en utilisant des Matchers.

La syntaxe est : **assertThat(message, actual, matcher)**

Exemples :

*assertThat*(2, *is*(*equalTo*(2)));

Pour des collections :

*assertThat*(listeDiplomes, *hasItems*(etudiantEncSlam,etudiantEncSisr));

Vérifie si la collection listeDiplomes contient les items etudiantEncSlam et etudiantEncSisr

*assertThat*(listeDiplomes, not(*hasItems*(etudiantEncSlam,etudiantEncSisr)));

Vérifie si la collection listeDiplomes NE contient PAS les items etudiantEncSlam et etudiantEncSisr

*assertThat*(« Liste pas vide attention »,listeDiplomes, IsEmptyCollection.*empty*());

Vérifie si la collection listeDiplomes est vide.

*assertThat*(« Liste pas vide attention »,listeDiplomes, IsCollectionWithSize.*hasSize*(2));

Vérifie si la collection listeDiplomes contient 2 éléments.

## Evolution et ajout d’une classe Manager

Le chef de projet de la société LOGIMOUVE demande une évolution de l’application métier pour prendre en compte le fait que les salariés sont gérés par un Manager.

Ce Manager est lui-même un salarié, qui gère une liste de salariés dont il est responsable.

L’architecte de l’application a déjà défini une interface Management avec une méthode *decideConges* qu’il s’agit d’implémenter dans la classe Manager.

Cette méthode prend en paramètre une liste de Salarie (la liste peut n’en contenir qu’un) et une décision et, si la décision est un refus, le motif de cette décision.

Si le salarié fait partie des salariés dont est responsable le Manager, alors la méthode va rechercher toutes les demandes d’absence associées au salarié en question, dont l’état est encore **en attente**, et leur affecter l’état et éventuellement le motif correspondant, passés en paramètre.