|  |  |
| --- | --- |
| **BTS SERVICES INFORMATIQUES AUX ORGANISATIONS** | **SESSION 2019** |

E5SL : PRODUCTION ET FOURNITURE DE SERVICES

|  |  |
| --- | --- |
| **Durée : 4 heures** |  |

CAS SUPMASTER

Ce sujet comporte 12 pages dont 10 pages de documentation.

La candidate ou le candidat est invité-e à vérifier qu’il est en possession d’un sujet complet.

*Aucun matériel ni document autorisé*

**Ressources documentaires :**

1. Présentation de la base de données
2. Extrait du diagramme des classes métier
3. Description détaillée des classes métier (extrait)
4. Utilisation d’une collection
5. Extrait du programme *RappVisite*
6. Demande de modification PO\_14
7. Début test unitaire

## Barème

|  |  |  |
| --- | --- | --- |
| Mission 1 | Suivi des contrats de travail des apprentis | 50 points |
| Mission 2 | Suivi des visites de prospection | 50 points |
|  | Total | 100 points |

**Présentation du contexte**

#### L'organisation cliente : le centre de formation d'apprentis SupMaster

SupMaster est un centre de formation d'apprentis (CFA) préparant des jeunes âgés de 18 à 25 ans au diplôme d'ingénieur. Établissement de formation professionnelle par alternance placé sous la tutelle du ministère de l'Education nationale, SupMaster est investi d'une mission de service public dont la création, le fonctionnement et les attributions sont très précisément décrits dans le livre I du code du travail. C'est au titre de cette mission de service public que le CFA perçoit de l'argent public qui couvre, en partie, ses frais de fonctionnement pédagogique.

SupMaster forme des chefs de projets (bac+5) alliant compétences techniques et managériales. Le centre couvre deux domaines : les systèmes d’information (cursus IM pour Informaticien Manager) et le multimédia (cursus MM pour Media Manager). L’aspect transversal des formations joue un rôle d’accélérateur d’employabilité pour des apprentis recrutés à bac+2. Les formations du CFA sont reconnues au niveau 1 par le RNCP (répertoire national des certifications professionnelles). SupMaster forme ses apprentis dans le cadre de contrats d’alternance pluriannuels entre l’étudiant, le CFA et une entreprise. Le principe des études consiste à effectuer une période d’un mois dans un établissement de formation SupMaster et une période d’un mois dans l’organisation.

Depuis sa création à Marseille en 2002, le centre de formation a choisi une stratégie de diversification géographique en s’implantant dans de grandes villes de France (Lyon, Lille, Paris, Strasbourg, Rennes et Bordeaux). Il forme aujourd’hui environ 300 étudiants, possède 25 salariés à temps plein, auxquels il faut ajouter 70 intervenants extérieurs qui réalisent différents enseignements dans les cursus proposés.

#### L'entreprise prestataire de services

SIForm est une entreprise de services du numérique (ESN) marseillaise implantée à proximité du centre de formation. Elle réalise des développements auprès de clients qu’elle démarche, ou pour ses besoins propres.

Elle compte un effectif de 6 personnes à temps plein qui interviennent dans tous les domaines de l’informatique d’aujourd’hui, et plus particulièrement les développements Web et mobiles.

#### Problématique du projet

Le choix exclusif de l’alternance pour tous les étudiants impose un travail important sur la reconnaissance de SupMaster par les organisations (entreprises, administrations, ou encore associations).) susceptibles de fournir des contrats aux étudiants. Pour maintenir sa place dans le tissu concurrentiel des écoles du domaine des technologies de l’information et de la communication, le directeur général, Léopold C. a décidé de développer une application destinée à améliorer la notoriété du centre. Compte tenu de la stratégie de diversification géographique, les organisations situées dans les régions couvertes par chaque établissement du centre constituent la cible principale de l’action.

Ce projet est confié à SIForm. L'ESN aura pour rôle d'adapter l'existant aux exigences nouvelles de la direction générale. La proximité géographique des deux structures facilite la communication et le contrôle des travaux. Fraichement embauché-e par SIForm, vous intervenez dans plusieurs missions mises en place par l'ESN dans le cadre de ce projet.**Mission 1 : Suivi des contrats de travail des apprentis**

*Document(s) à utiliser : 1*

Pour suivre ses promotions, SupMaster utilise une application qui s’appuie sur une base de données nommée *Supmaster* quistocke l'ensemble des informations relatives aux étudiants inscrits dans un cursus.

Lors de la mise en place de cette base de données, les contrats en cours ont été intégrés mais les contrats terminés ne l’ont pas été. C’est pourquoi un attribut *nbContratsOrg* a été ajouté dans la table Organisation. On s’est contenté de saisir le nombre de contrats ayant été signés avec chaque organisation. Cette information est mise à jour à l’aide du déclencheur (*trigger*) suivant :

CREATE TRIGGER SUPMASTER.majNbContrats

ON SUPMASTER.CONTRAT AFTER INSERT AS

BEGIN

UPDATE SUPMASTER.ORGANISATION

SET NBCONTRATSORG = NBCONTRATSORG +1

WHERE ID = (SELECT IDORG FROM INSERTED )

END

Afin de connaître le nombre de contrats que signe un étudiant lors de son passage au centre, la donnée *nbContratsEtud* vient d'être ajoutée dans la table Etudiant.

Le directeur souhaite effectuer un suivi des contrats. Pour cela, il a besoin :

* d'une liste, indiquant le nom et la ville de l'organisation, le nom et le prénom de l'étudiant ainsi que la date du contrat. Elle sera triée par nom d'organisation et par date de contrat.
* connaître la liste des étudiants sans contrat actuellement, afin de les radier ou de leur trouver un contrat rapidement : nom, prénom, code et libellé cursus, code établissement
* d’une liste des organisations ayant signé plus de 3 contrats (quelle que soit la période) indiquant le nom de l’organisation et le nombre de contrats.

### Votre mission

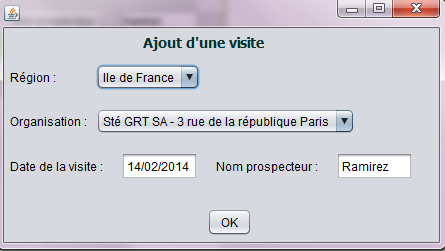
1. Mettre à jour la donnée nbContratsEtud de la table SUPMASTER.ETUDIANT, pour qu’elle reflète la réalité
2. Modifier le code du déclencheur pour mettre à jour la donnée nbContratsEtud lorsqu’un nouveau contrat est ajouté. Vérifier le code en ajoutant un nouveau contrat

1. Écrire les requêtes permettant d'obtenir les trois listes souhaitées.

**Mission 2 : Suivi des visites de prospection**

*Document(s) à utiliser : 2,3, 4, 5, 6, 7 et 8*

*IMPORTANT : la candidate ou le candidat doit présenter les éléments de code à l’aide du langage de programmation Java sous Eclipse.*

Pour prospecter les organisations, SupMaster emploie une personne par établissement. Pour faciliter le travail de ce prospecteur, une application java *RappVisite* est en cours d’écriture. Elle permettra notamment la saisie d’une visite par le prospecteur d’un centre SupMaster. L'exemple ci-contre, consiste à enregistrer la première visite réalisée auprès de l’organisation *GRT SA* située dans la région *Ile de France*, par Monsieur Ramirez, prospecteur de l’établissement *SupMaster-Paris*.

***T***rois points sont à traiter :

* À la 1ère exécution du programme, le message d'erreur suivant est apparu :

Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException

at supmaster.Organisation.ajoutVisite(Organisation.java:19)

at supmaster.RappVisite$1.actionPerformed(RappVisite.java:110)

* Le responsable du projet a constaté une erreur de conception dans le programme *RappVisite* et l'a formalisée dans la *demande de modification* *PO\_14*. Un de vos collègues a commencé à réfléchir à ce problème. Il a complété la *demande de modification* en précisant les étapes de l’implémentation technique.
* Pour améliorer le suivi des visites, le responsable des prospections souhaite que chaque prospecteur dispose de statistiques concernant son établissement. Il serait entre autres intéressant de connaitre, pour chaque région couverte par l’établissement, l’indicateur de couverture des visites qui est calculé ainsi :

nombre d’organisations de la région déjà visitées au moins une fois divisé par nombre total d’organisations de la région

Il semblerait qu’un de vos collègues ait d’ailleurs prévu d’effectuer un test unitaire de cette méthode, qu’il ait commencé (cf document 7).

### Votre mission

1. Expliquer la cause du message d'erreur apparu à la première exécution du programme *RappVisite* et donner la solution permettant d'y remédier.
2. Coder les différentes étapes permettant de réaliser la modification demandée dans la fiche PO\_14.
3. Coder une solution permettant de calculer l’indicateur de couverture des visites d’une région.
4. Compléter le test unitaire pour la solution de l’étape 6 (cf document 7. On suppose que l’indicateur implémenté à l’étape 6 est effectué via une méthode de la classe Region)

**Ressources documentaires**

# Présentation de la base de données

**Modélisation conceptuelle**

***La modélisation conceptuelle est volontairement fournie dans les deux représentations les plus courantes.***

|  |
| --- |
| **Diagramme de classes** |
|  |

**Schéma relationnel**

**Cursus (id, libelle)**

*Clé primaire : id*

*/\* Le libellé d’un cursus correspond à un choix de formation offert par le centre associé à l’année. Exemple : le cursus IM1 correspond à la 1ère année de formation d’Informaticien Manager. Tous les cursus sont disponibles au sein de chaque établissement de l’institut. \*/*

**Etudiant (id, nom, prenom, adresse, ville, nbContratsEtud, idEtablissement, idCursus)**

*Clé primaire : id*

*Clé étrangère : idCursus en référence à id de Cursus*

*/\* La donnée idEtablissement contient le code de l’établissement fréquenté par l’étudiant. Exemple : "mar" pour "Marseille" \*/*

**Activite (codeApe, libelle)**

*Clé primaire : codeApe /\* code correspondant à un secteur d’activité, défini par l’INSEE \*/*

**Organisation (id, nom, adresse, ville, nbContratsOrg, codeApeActivite)**

*Clé primaire : id*

*Clé étrangère : codeApeActivite en référence à codeApe de Activite*

*/\* La donnée nbContratsOrg contient le nombre de contrats signés par l’organisation \*/*

**Contrat (id, dateContrat, caractéristiques, dateFin, idOrg, idEtudiant)**

*Clé primaire : id*

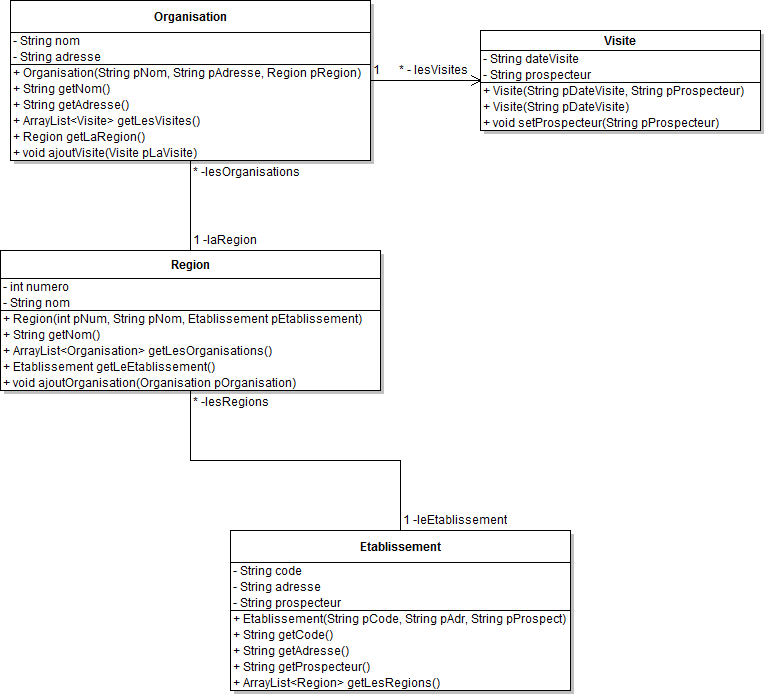
*Clé étrangère : idOrg en référence à id de Organisation*

*Clé étrangère : idEtudiant en référence à id de Etudiant*

*/\* La table Contrat mémorise les informations sur les contrats signés par les étudiants et les organisations. Il y a, pour chacun, une date de signature (attribut dateContrat) et une date de fin (attribut dateFin) qui n’est pas renseignée pour les contrats en cours\*/*

# Extrait du diagramme des classes métier

La description des classes est fournie ici à un niveau d’abstraction intermédiaire, suffisant pour la compréhension des informations principales à gérer.



# Description détaillée des classes métier (également fournies en code)

Classe REGION

**package** supmaster;

**import** java.util.ArrayList;

**public** **class** Region {

**private** **int** numero;

**private** String nom;

**private** Etablissement leEtablissement; // Etablissement de Supmaster couvrant la région

// Collection des organisations prospectées ou susceptibles de l’être dans la région.

**private** ArrayList <Organisation> lesOrganisations;

**public** Region(**int** pNum, String pNom, Etablissement pEtablissement)

{

**this**.numero = pNum ;

**this**.nom = pNom ;

**this**.leEtablissement = pEtablissement;

}

**public** **int** getNumero() {

**return** numero;

}

**public** String getNom() {

**return** nom;

}

**public** Etablissement getLeEtablissement() {

**return** leEtablissement;

}

**public** ArrayList<Organisation> getLesOrganisations() {

**return** lesOrganisations;

}

**public** **void** ajoutOrganisation(Organisation pOrganisation)

{

lesOrganisations.add(pOrganisation) ;

}

}

Classe ORGANISATION

**package** supmaster;

**import** java.util.ArrayList;

**public** **class** Organisation {

**private** String nom;

**private** String adresse;

**private** ArrayList<Visite> lesVisites;

**private** Region laRegion;

**public** Organisation(String pNom, String pAdresse, Region pRegion) {

**this**.nom = pNom ;

**this**.adresse = pAdresse ;

**this**.laRegion = pRegion ;

}

**public** String getNom() {**return** nom; }

**public** String getAdresse() {**return** adresse; }

**public** ArrayList<Visite> getLesVisites() { **return** lesVisites; }

**public** Region getLaRegion() { **return** laRegion; }

**public** **void** ajoutVisite(Visite pLaVisite) {

lesVisites.add(pLaVisite) ; }

}

Classe ETABLISSEMENT

**package** supmaster;

**import** java.util.ArrayList;

**public** **class** Etablissement {

**private** String code;

**private** String adresse;

**private** String prospecteur;

**private** ArrayList<Region> lesRegions; // zone géographique couverte par l’établissement

**public** Etablissement(String pCode, String pAdr, String pProspect)

{

**this**.code = pCode;

**this**.adresse = pAdr;

**this**.prospecteur = pProspect ;

}

**public** String getCode() { **return** code; }

**public** String getAdresse() { **return** adresse; }

**public** String getProspecteur() { **return** prospecteur; }

**public** ArrayList<Region> getLesRegions() { **return** lesRegions; }

}

Classe VISITE

**package** supmaster;

**public** **class** Visite {

**private** String dateVisite;

**private** String prospecteur;

// Constructeur qui initialise les deux attributs avec les deux paramètres

**public** Visite(String pDateVisite, String pProspecteur) {

**this**.dateVisite = pDateVisite ;

**this**.prospecteur = pProspecteur ;

}

**public** Visite(String pDateVisite) { **this**.dateVisite = pDateVisite ; } //constructeur qui initialise uniquement la date

**public** **void** setProspecteur(String pProspecteur) { **this**.prospecteur = pProspecteur ;} // renseigne l’attribut prospecteur

}

# Utilisation d’une collection

*L’exemple ci-dessous permet de manipuler une collection de chaînes de caractères. Le principe est le même quel que soit le type des éléments.*

ArrayList<String> mesChaines; // déclaration d'une collection de chaînes de caractères

mesChaines = new ArrayList<String>(); // instanciation de la collection

mesChaines.add("un"); // ajout d'une chaîne à la collection

mesChaines.add("deux");

mesChaines.add("trois");

System.out.println(mesChaines.size()); // affichage du nombre d'éléments de la collection

for (String uneChaine : mesChaines) { // parcours de la collection

System.out.println(uneChaine); // affichage de l'élément courant

}

mesChaines.remove(1); // suppression du 2ème élément (indice 1)

System.out.println(mesChaines.get(0)); // affichage du 1er élément (indice 0)

# Extrait du programme *RappVisite*

Extrait lignes 102 à 114

**public** **void** actionPerformed(ActionEvent e) {

Etablissement unEtablissement = **new** Etablissement("FRA","Inconnu","Directeur") ;

Region uneRegion = **new** Region(1,"France", unEtablissement) ;

Organisation uneOrga = **new** Organisation(organisationList.getSelectedValue().toString(),

" inconnue ",uneRegion) ;

Visite uneVisite = **new** Visite( dateInput.getText(),nomInput.getText());

uneOrga.ajoutVisite(uneVisite);

errorText.append(uneOrga.toString());

}

# Demande de modification PO\_14

|  |
| --- |
| Projet : Prospection des Organisations (PO) Émise le : 26/02/2014  Demande de modification n° : PO\_14 |
| **Description de la modification demandée**  Nom :....Gandri Jacques Fonction : Responsable du projet  Niveau d’importance : Faible □  Majeur ■  Bloquant □  *Fonctionnalité concernée* : saisie des visites réalisées par les prospecteurs.  *Description :* La saisie du nom du prospecteur dans la fenêtre du programme *RappVisite* pourrait induire des incohérences avec l’attribut *prospecteur* de la classe Etablissement. La solution consiste à ne demander que la saisie de la date dans l'interface de saisie. |
| **Implémentation technique :**   1. Modification du formulaire de saisie d’une visite (programme RappVisite) : saisie de la date seule (la visite sera instanciée avec le constructeur paramétré qui instancie la visite uniquement avec la date) 2. Réécriture de la méthode *AjoutVisite()* de la classe *Organisation* avec la récupération du nom du prospecteur grâce à l’attribut *prospecteur* de la classe *Etablissement*, *m*ise à jour de l'attribut *prospecteur* de l'objet *Visite* puis ajout de la visite à la collection. 3. Affichage dans la fenêtre de message d’erreur / info de l’application (le JTextArea errorText) de l’instance Organisation en cours (à laquelle on vient d’ajouter une visite) |

# Morceau de test unitaire pour l’indicateur du taux de couverture

Region uneRegion ;

@BeforeEach

**void** setUp() **throws** Exception {

/\*\*\*\* Création d'un établissement pour le Test \*\*\*/

Etablissement unEtablissement = **new** Etablissement("Paris","Inconnu","Robert") ;

/\*\*\*\* Création d'une région pour le Test, associée à l'établissement (Paris) \*\*\*/

uneRegion = **new** Region(1,"Ile de France", unEtablissement) ;

/\*\*\* Création d'une 1ère organisation \*\*\*\*/

Organisation uneOrga = **new** Organisation("SUADEO","inconnue",uneRegion) ;

/\*\*\* Ajout d'une visite pour cette 1ère organisation \*\*\*\*/

Visite uneVisite = **new** Visite( "2018-10-11");

uneOrga.ajoutVisite(uneVisite);

uneRegion.ajoutOrganisation(uneOrga);

Organisation deuxOrga = **new** Organisation("MUNIVIE","inconnue",uneRegion) ;

uneRegion.ajoutOrganisation(deuxOrga);

Organisation troisOrga = **new** Organisation("Umanis","inconnue",uneRegion) ;

uneRegion.ajoutOrganisation(troisOrga);

}

# Exemples 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.