Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'ÉCONOMIE
ET DES FINANCES

MINISTÈRE DE L'ACTION
ET DES COMPTES PUBLICS

secrétariat général DES MINISTÈRES ÉCONOMIQUES ET FINANCIERS

EXAMEN PROFESSIONNEL DE VERIFICATION D'APTITUDES AUX FONCTIONS DE PROGRAMMEUR

1re session – 2020

ÉPREUVE ÉCRITE D'ADMISSIBILITÉ DU 14 JANVIER 2020

ÉTABLISSEMENT D'UN ALGORITHME DANS UN LANGAGE CHOISI PAR LE CANDIDAT
(JAVA, PHP, VB/ASP.NET, PYTHON, PERL, JAVASCRIPT, C++, OBJECTIVE C, SWIFT)

(Durée : 5 heures)

REMARQUES IMPORTANTES :

- L'usage de calculatrices, de règles de calcul, de tables de logarithmes et de tout document est strictement interdit.
- Les copies doivent être rigoureusement anonymes et ne comporter aucun signe distinctif ni signature, même fictive, sous peine de nullité.
- Le candidat s'assurera, à l'aide de la pagination, qu'il détient un sujet complet (le sujet comporte 11 pages).

TOUTE NOTE INFÉRIEURE À 10 SUR 20
EST ÉLIMINATOIRE

Présentation

La Super Société d'Informatique Innovante (SSII)

Fondée dans les Ardennes en 1965, SSII est présente dans 158 pays et compte plus de 120 000 collaborateurs. Elle propose un ensemble de prestations couvrant la plupart des aspects des technologies de l'information et de la communication. Elle compte de nombreux clients prestigieux comme la Banque Centrale de Syldavie.

En tant qu'Entreprise de Services du Numérique (ESN), elle met à disposition de ses clients, pour des missions définies, des équipes de développeurs. Ces missions ont un début et une fin déterminés et nécessitent un certain nombre de compétences. Pour qu'une mission puisse être réalisée à bien, il faut que chacune des compétences nécessaires à la mission soit maîtrisée par au moins un des développeurs participant à la mission et que les développeurs soient disponibles pendant les dates de la mission. C'est-à-dire qu'un même développeur ne peut pas travailler sur plusieurs missions en même temps. Pour travailler sur une mission, un développeur doit aussi faire partie de l'entreprise, il doit avoir été embauché et ne pas avoir quitté l'entreprise. Chaque mission est négociée par un unique commercial, pour le compte d'une unique entreprise. Chaque entreprise appartient à un unique secteur d'activité.

Afin de faciliter la gestion des missions et la constitution des équipes de développeurs, SSII a mis en place une application. Sa modélisation est présentée en annexe 1.

Tarification

Le prix total d'une mission correspond à la somme des facturations des développeurs impliqués dans la mission pondérée par le taux de marge associé au commercial ayant négocié la mission. On considère que chaque développeur travaille tous les jours compris entre le début et la fin de la mission **inclus.** La date de référence utilisée pour facturer la prestation d'un développeur est la date de début de la mission. C'est-à-dire que pour une mission donnée, l'ancienneté de chaque développeur correspond à la différence entre sa date d'embauche et la date de début de la mission. Une mission implique un ou plusieurs développeurs et un unique commercial. Les développeurs travaillent du premier au dernier jour de la mission, week-end inclus. Cependant les employés de SSII sont heureux car un toboggan est mis à leur disposition pour se rendre au réfectoire.

Les taux journaliers (facturation d'un développeur pour une journée) dépendent uniquement de l'ancienneté des développeurs. On considérera qu'une année fait 365 jours. Seul le nombre d'années entières compte. Les taux journaliers sont calculés selon le barème suivant :

Titre	Ancienneté	Facturation
Junior	<= 3 ans	200 + 20 euros par année d'expérience
Senior	> 3 ans et <=6 ans	300 euros + 20 euros par année d'expérience
Expert	> 6 ans	500 euros + 20 euros par année d'expérience

Par exemple, le taux journalier d'un développeur ayant douze ans, deux mois et six jours d'expérience est de 500 + 12 * 20 = 740 euros.

Si deux développeurs ayant respectivement 1 et 5 ans d'expérience participent à une mission commençant le 01/10/2019 et finissant le 20/10/2019 et qu'elle est négociée par un commercial avec un taux de marge de 1,2, le prix de facturation sera de :

$$(200 + 20 + 300 + 100) \times 20 \times 1{,}2 = 13\,440\ euros$$

Questions

- Dans les questions suivantes, on demande d'écrire des méthodes et des fonctions. Lorsqu'il faut écrire une méthode, la classe dans laquelle elle doit figurer est précisée. Lorsqu'il faut écrire une fonction, on considère que la fonction est écrite en dehors de toute classe[1]. Les implémentations sont à fournir en pseudo-code.
- Les questions peuvent être traitées indépendamment. On pourra utiliser des fonctions ou des méthodes définies dans les questions précédentes même si on n'a pas traité la question.
- L'annexe 1 présente le diagramme de classes de l'application.
- L'annexe 2 présente chacune des classes métier une par une.
- L'annexe 3 fournit un certain nombre de classes utilitaires qui **devront être utilisées** pour répondre aux questions.
- Les attributs et méthodes sont accompagnés de documentation au format pseudoCodeDoc, c'est-à-dire du texte en italique. Pour chacun des attributs de visibilité "privé", on supposera que des accesseurs publics sont définis. Par exemple, pour la classe Employe, on pourra notamment utiliser les méthodes getNom(): Chaine et setNom(nom: Chaine).
- On considérera que les listes et les chaînes de caractères (classes Liste<T> et Chaine) sont indexées à partir de 0.

1) Dans la classe Developpeur, écrire la méthode **tauxJournalier(dateReference Date): flottant** qui calcule et retourne le prix de facturation d'un développeur pour une journée.

2) Dans la classe Mission, écrire la méthode **coutTotal(): flottant** calculant le coût total d'une mission.

3) Dans la classe Developpeur, écrire la méthode **peutParticiperAMission(mission: Mission): booleen** qui retourne :
   - vrai si le développeur est disponible pendant toute la durée de la mission
   - faux sinon

4) Écrire une fonction **moyenneCA(codeAPE: chaine): flottant**, calculant le tarif moyen des missions d'un secteur d'activité donné. On devra faire un appel à la base de données.

5) Dans la classe Mission, écrire une méthode **equipeValide(): booleen** qui retourne vrai si pour chacune des compétences nécessaires à la mission, un membre de l'équipe au moins possède cette compétence, et chacun des membres est disponible.

1 Bien que cette possibilité ne soit pas offerte par tous les langages de programmation objet.

6) On cherche à composer une équipe pour une mission donnée. A cette fin, on va essayer de composer une équipe possédant toutes les compétences requises pour la mission. On va procéder par énumération. On veut donc tester toutes les sous-parties de l'ensemble. On remarquera dans un premier temps que chaque sous-partie d'un ensemble de n éléments correspond à un et un seul nombre binaire à n chiffres. Chaque chiffre indique si le i-ème élément est présent ou non.
Par exemple, si on dispose de trois développeurs, Jean, Marie et Paul, les possibilités sont les suivantes :

Nombre binaire	Équipe correspondante
[0,0,0]	{}
[0,0,1]	{Paul}
[0,1,0]	{Marie}
[0,1,1]	{Marie, Paul}
[1,0,0]	{Jean}
[1,0,1]	{Jean, Paul}
[1,1,0]	{Jean, Marie}
[1,1,1]	{Jean, Marie, Paul}

a) Écrire une fonction **incrementer(nombre: Liste<entier>): Liste<entier>**. Elle retournera la liste représentant nombre+1 en binaire, et null si la capacité de la liste n'est pas suffisante.
Par exemple incrementer([0,0,0]) = [0,0,1]
incrementer([1,0,1]) = [1,1,0]
incrementer([1,1,1]) = null

b) Écrire une fonction **equipeLaMoinsChere(mission: Mission): liste<Developpeur>**, retournant la liste des Developpeur possédant toutes les compétences requises et disponibles pour participer à la Mission, et pour laquelle le coût est minimum. Si une telle liste n'existe pas, la méthode retourne null. On pourra utiliser les méthodes des questions 5 et 6-a.

c) Combien de cas faudra-t-il énumérer si l'on a : 2, 3, 4, 5, 10 ou n développeurs ? Un tel algorithme est-il raisonnable si l'on a 120 000 collaborateurs ? Justifier votre réponse.

7) Suite à une fuite de données, le directeur de la sécurité décide de mettre en place un système de chiffrement. Après avoir lu un livre[2] de cryptographie jusqu'à la onzième page, il propose de mettre en place un chiffrement selon la méthode de Vigenère. On dispose de deux chaînes de caractères en entrée : un message m et une clé c. Le chiffrement consiste à décaler chaque caractère du message m selon la position du caractère correspondant dans la clé C. Si la clé est plus courte que le message, elle est répétée autant de fois que nécessaire pour que les longueurs coïncident.
Par exemple pour le message : "ce message est indecodable" et la clé "ficus", on obtient :

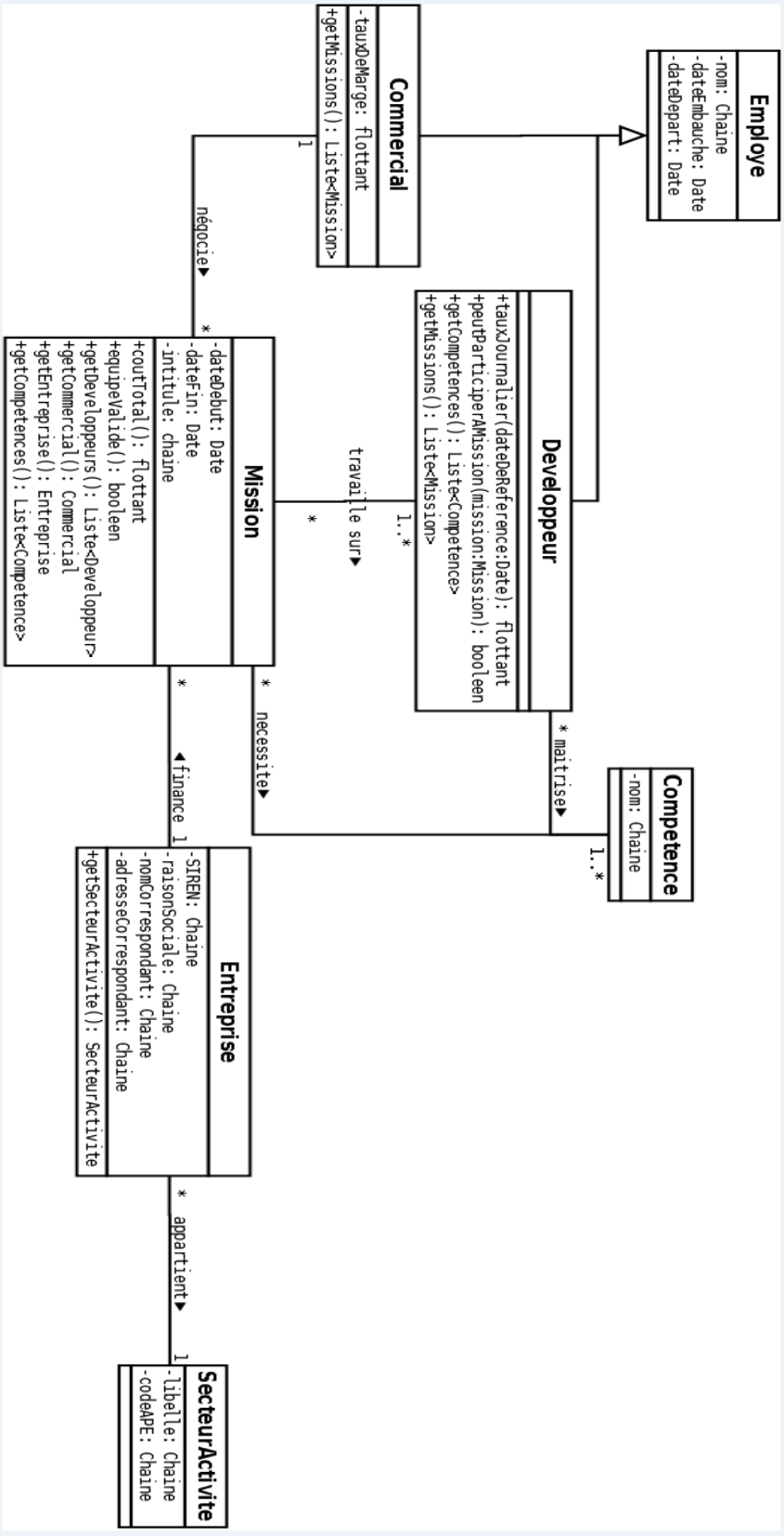
Message	c	e		m	e	s	s	a	g	e		e	s	t		i	n	d	e	c	o	d	a	b	l	e
Rang	3	5		14	5	19	19	1	7	5		5	19	20		9	14	4	5	3	15	4	1	2	12	5
Clé	f	i		c	u	s	f	i	c	u		s	f	i		c	u	s	f	i	c	u	s	f	i	c
Rang	6	9		3	21	19	6	9	3	21		19	6	9		3	21	19	6	9	3	21	19	6	9	3
Somme	9	14		17	26	12	25	10	10	26		24	25	3		12	9	23	11	12	18	25	20	8	21	8
Message chiffré	i	n		p	z	l	y	j	j	z		x	y	c		j	i	w	k	l	r	y	t	h	u	h

Écrire la fonction **chiffrer(message: Chaine, cle: Chaine): Chaine** qui chiffre le message à l'aide de la méthode de Vigenère.

8) Traduire la question 5 dans le langage de votre choix.

2 *Les méthodes de chiffrement expliquées à mon chien, 1494. 317 p.*

Annexe 1 : diagramme de classes



Annexe 2 : descriptions des classes

Employe
- nom: Chaine *Nom de l'employé.* - dateEmbauche: Date *Date d'embauche de l'employé.* - dateDepart: Date *Date à laquelle l'employé a quitté l'entreprise. Vaut null si la date de départ de l'employé n'est pas encore connue.*

Developpeur
+ tauxJournalier(dateDeReference: Date): flottant *Calcule le taux journalier d'un développeur à la date dateDeReference. Ce taux dépend de son ancienneté (nombre d'années de différence entre sa dateEmbauche et la dateDeReference).* + peutParticiperAMission(mission: Mission): booleen *Retourne vrai si le développeur est disponible pour travailler sur la mission, c'est-à-dire qu'il n'est pas occupé par une autre mission et qu'il fait bien partie de l'entreprise (il a bien été embauché lorsque la mission commence et il n'a pas été licencié lorsqu'elle se termine). Retourne faux sinon.* + getComptences(): Liste<Competence> *Retourne la liste des compétences du développeur.* + getMissions():Liste<Mission> *Retourne la liste des missions auxquelles participe ou a participé le développeur.*

Commercial
- tauxDeMarge: flottant *Le taux de marge est un nombre à virgule flottante compris entre 1 et 2. Il sert à calculer le prix de facturation d'un projet. Pour obtenir ce prix, il faut multiplier la somme des tarifs de facturation des développeurs par le taux de marge. Il ne s'agit pas au sens strict d'un taux de marge mais plutôt d'un coefficient. Par exemple, pour une marge de 20 %, tauxDeMarge vaudrait 1,2.*
+ getMissions(): Liste<Mission> *Retourne la liste des missions négociées par le commercial.*

Mission
- dateDebut: Date *Date de début de la mission.* - dateFin: Date *Date de fin de la mission.* - intitule: Chaine *Intitule de la mission.*
+ coutTotal(): flottant *Calcule le prix de facturation d'une mission, c'est-à-dire le prix de facturation des développeurs multiplié par le taux de marge associé au commercial.* + equipeValide(): booleen *Retourne vrai si pour chacune des compétences associées à la mission, au moins un des développeurs associés à la mission la maîtrise, et si, de plus, chacun des développeurs associé à la mission est disponible pour travailler sur cette mission.* + getDeveloppeurs(): Liste<Developpeur> *Retourne la liste des développeurs associés à la mission.* + getCommercial(): Commercial *Retourne le commercial ayant négocié la mission.* + getEntreprise(): Entreprise *Retourne l'entreprise* + getCompetences() Liste<Competence> *Retourne la liste des compétences nécessaires pour accomplir la mission.*

Entreprise
- SIREN: Chaine *Code d'identification unique de l'entreprise dans le répertoire des entreprises SIRENE. Code de neuf caractères.* - raisonSociale: Chaine *Raison sociale de l'entreprise.* - nomCorrespondant: Chaine *Nom de la personne à contacter dans l'entreprise.* -adresseCorrespondant: Chaine *Adresse de la personne à contacter dans l'entreprise.*
+ getSecteurActivite(): secteurActivite *Retourne le secteur d'activité de l'entreprise.*

SecteurActivite
- libelle: Chaine *Libellé du secteur d'activité.* - codeAPE: Chaine *Code d'Activité Principale Exercée.*

Annexe 3 : classes utilitaires

BaseDeDonnees
+ baseDeDonnees() *Constructeur* + connecter() : Booléen *Permet de se connecter à la base de données, vrai si réussi* + deconnecter() : Booléen *Permet de se déconnecter de la base de données, vrai si réussi.* + listeMissions() : Liste<Missions> *Retourne la liste des Mission présentes dans la base de données.* + listeDeveloppeurs() : Liste<Developpeur> *Retourne la liste des Developpeur présents dans la base de données.*

Date
+ Date(jour: entier, mois: entier, annee: entier) *Constructeur retournant la date jour/mois/annee.* + difference(d2: Date): entier *Retourne le nombre de jours entre d1 (l'objet sur lequel on appelle la méthode) et d2. Le résultat est négatif si d2 est postérieur à d1.* *Exemple :* *d1, d2: Date;* *d1 = new Date(12,2,2020);* *d2 = new Date(1,2,2020);* *d1.difference(d2); // L'évaluation de cette expression renvoie 11* *d2.difference(d1);// -11* + getAnnee() : entier *Retourne l'année.* + getMois(): entier *Retourne le mois.* + getJour(): entier *Retourne le jour.*

Le type générique Liste<T> peut être utilisé aussi bien avec des types primitifs qu'avec des classes.
Ainsi, les deux déclarations suivantes sont valides :

- listeEntiers: Liste<entier>;
- listeProgrammeurs: Liste<Programmeur>;

Liste<T>
+ Liste<T>() *constructeur : construit une liste vide* + Liste<T>(n: entier) *Construit une liste de n éléments valant 0 si T est un type primitif ou null sinon* + ajouter(unObjet : T) *Ajoute un nouvel objet à la fin du tableau* + ajouter(unObjet : T, position : Entier) *Insère un objet à la position passée en paramètre* + get(position : Entier) : T *Retourne l'objet situé à la position passée en paramètre* + set(unObjet : T, position : Entier) *Affecte unObjet à l'élément* + [] : T *On peut utiliser les opérateurs [] de façon équivalente aux accesseurs et mutateurs. Ainsi, si l'on dispose d'une variable maListeEntiers de type Liste<Entier> :* *maListe[1] équivaut à maListe.get(1)* *maListe[1] = 4 équivaut à maListe.set(4,1)* + enlever(position : Entier) *Supprime l'objet situé à la position passée en paramètre* + taille() : Entier *Retourne le nombre d'éléments de la Liste<T>* + suivant() / precedent() : T *Accède à l'élément suivant / précédant l'élément sur lequel on est positionné dans le tableau. Retourne null quand on cherche un élément hors du tableau, au début ou à la fin* + position() : Entier *Donne l'indice de l'élément sur lequel on est positionné. Le premier élément du tableau a pour indice 0.*

Les chaînes de caractères sont des suites de caractères. Les caractères sont des entiers qui sont simplement interprétés différemment lors de l'affichage. On supposera la correspondance entier/caractère suivante : 'a' = 1, 'b' = 2, 'c' = 3..., 'z' = 26 et ' ' = 27. Le dernier caractère est le caractère espace. Les deux notations sont strictement équivalentes, ainsi, évaluer l'expression :
'a' == 1 renverrait *true*.

Chaine
+ egale(chaine: Chaine): booleen *Renvoie vrai si les deux chaînes sont égales, faux sinon.* + longueur(): entier *Renvoie le nombre de caractères de la chaîne.* + caractereA(position: entier) : entier *Retourne l'entier correspondant au caractère à la position spécifiée. Les Chaine sont indexées à partir de 0.* *Par exemple :* *"ace".caractereA (0) = 1* *"ace".caractereA (1) = 3* *"ace".caractereA (2) = 5* *Remarque : pour désigner un caractère, on peut utiliser leurs rangs, ou des littéraux de type 'a', 'b'.*