**EQUIPE: Khang Nguyen 300007277**

**Pascal M-P 300034507**

**Partie 1**

1. Apply
   1. checkAvailable(), classe Position

return le boolean du attribut isAvailable. Pour voir si la position est déjà rempli.

return: isAvailable

* 1. checkPreRequis(), classe Position

return le String preRequis qui est un string avec tous les prérequis, cela vas être comparé avec les attributs sur le CV pour voir si l’élève a les prérequis

return: preRequis

* 1. Application(Course[]), classe Application

constructeur pour application, créer l’application pour le course ou les courses, pas de return statements

paramètres: Courses[] - array des courses pour laquelle l’application se concerne

1. Check positions
   1. getCourses(), classe University

return un array de tous les courses qui sont en demande d’un TA

return: Courses [] array des courses offert à l’université

* 1. getRelevantCourses(Courses[]), classe Student

return un array de tous les courses pour laquelle l’étudiant a déjà pris les courses et a les préRequis

paramètre: Courses [] array des courses offert à l’université

return: Courses [] - array des courses potentiel pour TA

* 1. getAvailableCourses(Courses[]), classe Student

return un array de tous les courses dans laquelle l’étudiant peut appliqué, en utilisant checkAvailable pour voir si elles sont encore ouvert.

paramètre: Courses [] array des courses potentiel pour TA

return: Courses [] - array des courses valid pour application

1. Check applications
   1. getApplications(Position), classe Professor

return un array d'applications pour un position.

paramètre: Position

return: Application [] - array des applications pour la position

1. Create position
   1. checkPosition(), classe Course

check si la course a déjà un position de cet type disponible, si oui créer pas un position pour cette class (1 position posté mais plus que 1 personne peut être accepter, comme CO-OP), isAvailable va toujours être vrai lorsque tu créer la position

* 1. Position(hourlyRate, preRequis, description, postedDT, overDT, classification, numberOfPositions), classe Position

constructeur pour Position, créer un position

paramètre: hourlyRate - combien d’argent par heur

preRequis - les prerequis pour la position

description - description de la position

postedDT,OverDT - date et temps pour lorsque la position commence et fini

classification - classification de la position

numberOfPositions - nombre de positions ouverts de cet type

1. Pick TA
   1. checkPreRequis(), classe Position

return le String preRequis qui est un string avec tous les prérequis, cela vas être comparé avec les attributs sur le CV pour voir si l’élève a les prérequis

return: preRequis

* 1. addPosition(Student), classe Position

créer un lien entre l’étudiant et la position, ensuite ajoute 1 au nombre de position current, si ce nombre est égale au numberOfPositions set isAvailable to false

paramètre: Student - élève qui est accepté

* 1. addPosition(Position), classe Student

créer un lien entre l’étudiant et la position, si l’université veut un maximum sur le nombre de position par élève simplement enlever tous application de l’élève

paramètre: Position - position de l’élève

**Partie 2**

1. “Chaque application Java possède une seule instance de la classe Runtime qui permet à l'application de se connecter à l'environnement dans lequel l'application est en cours d'exécution. Le runtime actuel peut être obtenu à partir de la méthode getRuntime. Une application ne peut pas créer sa propre instance de cette classe.” Nous pouvons voir à partir de cet extrait du JavaDoc de java.lang.Runtime que il y a seulement un instance de classe runtime par application java, de plus une application ne peut pas créer un (autre) instance de java.lang.Runtime. Tout ceci montre que java.lang.Runtime est un Singleton. Le fait que tous les méthodes sont publique et qu’il ne peut seulement avoir 1 instance montre cela.
2. “Définit une API de fabrique permettant aux applications d'obtenir un analyseur qui produit des arborescences d'objets DOM à partir de documents XML.” Nous pouvons voir à partir de cet extrait du JavaDoc de javax.xml.parsers. DocumentBuilderFactory que elle créer des “DocumentsBuilder” qui a leurs tours créer des documents. Le fait que javax.xml.parsers.DocumentBuilderFactory créer des “factories” dit qu’elle est un Abstract Factory. Un abstract factory créer des factories qui a leur tours créer des objets, ceci peut être vues par la méthode newDocumentBuilder() qui créer un instance document builder qui créer des XML files. D’un façon plus général elle pourrait aussi être un Factory.
3. Pour java.io.OutputStreamWriter, nous pouvons voirs à travers du méthode “Write” que le java.io.OutputStreamWriter vas “override” le write function de java.io.writer. Nous pouvons voir qu’un class qui “override” un autre class pour mieux fonctionner est sous le patron de “Adapter”, cela est vue avec le fait que java.io.OutputStreamWriter “override” java.io.writer
4. “Interface implémentée par des objets souhaitant être notifiés au début et à la fin du traitement pour chaque phase standard du cycle de vie du traitement de la demande.” Nous pouvons voir à partir de cet extrait du JavaDoc de javax.faces.event.PhaseListener qu’elle a seulement des méthodes qui sont exécutés durant des phases d’un autre classe. Cela est un patron “Observer”, cela est en raison que les méthodes de phaselistener observe les changements des autres class, pour exemple, afterPhase() beforePhase et getPhaseId. Tous les méthodes précédent est seulement exécuter à partir de l’information d’un autre classe.
5. “Exprime une dépendance sur un EntityManager géré par le conteneur et son contexte de persistance associé.” Nous pouvons voir à partir de cet extrait du JavaDoc de javax.persistence.PersistenceContext qu’elle est dépendant sur EntityManager. javax.persistence.PersistenceContext est un implémentation de EntityManager et donc EntityManager a délégué ses tâches à javax.persistence.PersistenceContext. Donc javax.persistence.PersistenceContext est un patron “Proxy” ou un patron de délégation.

CONTRAINTS

constraints  
context RecomendationLetter  
 inv atLeastOne: Banana.allInstances()->size()>1  
  
context Student

inv ValidName:  
 self.name.size()>=1  
  
 inv UniqueNumber:  
 self.allInstances() -> forAll(e1,e2|((e1.stuNumber = e2.stuNumber) implies (e1 = e2)))

//Si les deux eleves on le meme student number, sa implique que ces le meme objet. Sinon pas bon

inv registered

self.allInstacnes() -> forAll(e1|self.name = e1.name) implies (self = e1)