Examen Java Modèle 5 - Corrigé

Session : ING1 - Juin 2008 - session 2 - CERGY / PAU

Thème : les réseaux sociaux

L'actualité du Web est très prolixe en ce moment sur les thèmes des réseaux de personne. Nous utilisons ce prétexte conjoncturel pour vous faire réfléchir sur les nécessités "programmatiques" de ce type de réseaux.

Le but du devoir est de modéliser un petit réseau de personnes dans lequel on peut retrouver le "circuit" permettant de négocier entre deux personnes "par relation". Vous construirez un réseau simple, définirez une façon de qualifier les relations (guidée par le sujet) puis élaborerez un algorithme de recherche d'un chemin qui permette de passer par des "amis" et des "bonnes recommandations".

La programmation doit être claire.

La programmation doit faire usage des collections, et ne pas réimplémenter des algorithmes triviaux.

Vous devez mettre en oeuvre toutes vos connaissances Java pour proposer des solutions aux questions posées.

COMMENTEZ ETJUSTIFIEZ VOS INTENTIONS.

Questions de cours POO (4 points)

A quoi sert une classe finale ?

Une classe finale permet de bloquer toute dérivation ultérieure. En bloquant la possibilité de dérivation, on bloque aussi la possibilité de redéfinir différemment les méthodes.

Pourquoi l'encapsulation très privée rallentit-elle le travail de développement ? Pouvez-vous mettre ceci en regard avec les avantages que cette pratique procure ?

Parce que tout accès à une information suppose de passer par un accesseur (méthode publique) qui n'est pas toujours écrite au départ. De plus, l'écroture des accesseurs es plus longue que l'écriture d'un simple accès à un membre.

L'avantage vient surtout du fait que refermer l'objet permet d'en protéger le fonctionnement contre des altérations non contrôlées des membres données. Fermer l'objet permet de garantir que son état interne reste cohérent.

Quel est le bénéfice de la programmation objet ?

Le bénéfice est énorme lorsqu'on se sert de très nombreux exemplaires de structure de données identiques (instances) dont on peut définir une fois pour toute le comportement (méthodes).

Le développement objet découpe (comme les types abstraits) le problème en petites unités plus faciles à concevoir séparément (diviser pour mieux régner). Chaque objet est un petit mécanisme indépendant dont on peut vérifier qu'il fonctionne bien (tests unitaires).

Toutes les classes d'une application Java ne sont pas nécessairement toujours "montées" en mémoire vive. Expliquez pourquoi.

Parce que Java est activé dans une machine virtuelle qui charge les classes dynamiquement (Class Loader). L'édition de liens est dynamique, contrairement au C, (ou réalisé à grands frais de complication dans les dlls). De ce fait, si une classe n'est jamais "utilisée" son code ne sera pas monté en mémoire. C'est notamment le cas quand la classe principale, celle du main, n'utilise que des interfaces ou des superclasses, sans savoir par quelles classes concrètes elles sont réalisées effectivement.

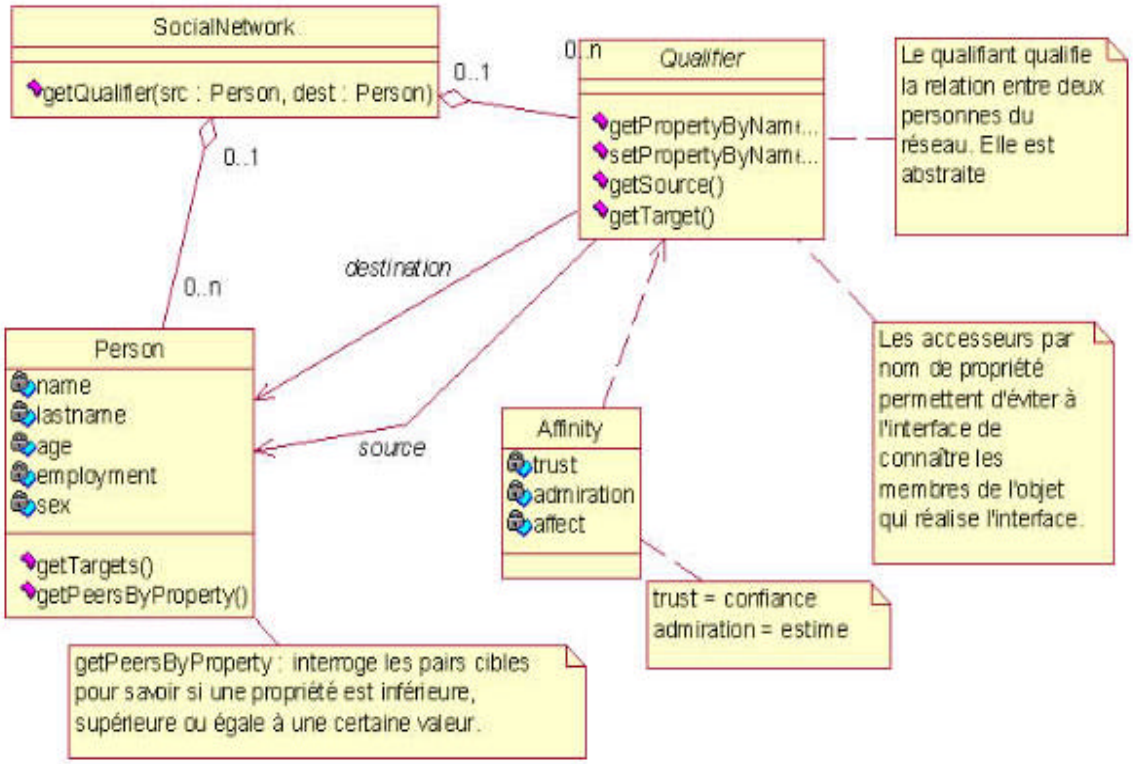
Conception de classes (6 points)

Un réseau social est composé d'individus en relation.

1. Déterminez la classe Personne représentant un individu du réseau. Fournissez entre trois et cinq champs descriptifs. La classe personne doit être totalement mutable.

2. Déterminez une façon de constituer un réseau de personnes. Identifiez (la ou) les collections nécessaires pour que :
    - Chaque personne puisse être liée à un nombre quelconque d'autres personnes (si une personne est liée à elle-même, on pourrait y coder une "estime de soi"!!).
    - Le réseau et ses relations ne CONNAISSENT PAS à priori quels sont les qualifications de ces relations.
    - Le réseau a un existence popre en tant qu'objet. Les mécanismes de collection doivent être intégrés PAR COMPOSITION.
    - Malgré le point précédent, nous savons que ces relations seront qualifiées par quelque chose.
    - Nous savons également que les qualifiant sont des propriétés, assimilables à des paires nom => valeur, les deux étant exprimables en texte. Vous devrez trouver quel objet pourrait répondre à ce besoin.
    - Utilisez une interface pour stocker l'instance qui qualifie la relation.
3. Constituez les classes sous forme de carcasses SANS ECRIRE LE CODE DES METHODES. Votre objectif est de bien décider des membres et des méthodes utiles pour manipuler ce modèle et de savoir le proposer en JAVA.
4. Le réseau doit être manipulable, c'est-à-dire que toutes les méthodes qui permette de le construire à partir d'un réseau vide doivent être présentes.
5. On désire qualifier les relations par une mesure d'affinité. Cette classe très simple stocke quelques qualificateurs quantifiés pour mesurer l'affinité entre deux personnes (Attention, penser que ces relations n'ont pas forcément la même valeur dans les deux sens !!) :
    - L'amitié affective (0 à 10)
    - L'estime professionnelle (0 à 10)
    - Le degré de confiance (0 à 10)

Pour vous aider, une transcription UML partielle a été faite de ce problème :



Vous devez évidement trouver comment enregistrer les "couples" de Affinity pour chaque "connexion" entre des personnes du réseau.

Barème :

- Explications et analyse du problème
- carcasse de la classe Personne

```
import java.util.Vector;


public class Person {
```

```
  protected String firstname;
  protected String lastname;
  protected int age;
  protected char gender;
  protected String nickname;

  public static final char MALE = 'M';
  public static final char FEMALE = 'F';

  // Constructors
  public Person(String fn, String ln, String nick, int age, char g){
   firstname = fn;
   lastname = ln;
   nickname = nick;
   this.age = age;
   gender = g;
  }

  // All accessors for mutability
public String getFirstname() {
 return firstname;
}

public void setFirstname(String firstname) {
 this.firstname = firstname;
}

public String getLastname() {
 return lastname;
}

public void setLastname(String lastname) {
 this.lastname = lastname;
}

public int getAge() {
 return age;
}

public void setAge(int age) {
 this.age = age;
}

public char getGender() {
 return gender;
}

public void setGender(char gender) {
 this.gender = gender;
}

public String getNickname() {
 return nickname;
}

public void setNickname(String nickname) {
 this.nickname = nickname;
}

// round 2

 public Person[] getTargets(){
 }

/**
 *
 * @param name
 * @param threshold
 * @return
 */
 public Person[] getPeersByProperty(String name, String threshold){
 }
```

```
  //round 3
  public String toString(){
  }
}
```

- carcasse de la classe Réseau

```
import java.util.ArrayList;

import networkexceptions.NotInPopulationException;


public class SocialNetwork {

 protected ArrayList<Person> population;
 protected ArrayList<Qualifier> relationships;

 public SocialNetwork(){
  population = new ArrayList<Person>();
  relationships = new ArrayList<Qualifier>();
 }

 /**
  * ensures single record
  */
 public void addPerson(Person p){
  if (!population.contains(p)){
   population.add(p);
  }
 }

 /**
  *
  */
 public void removePerson(Person p){
  if (population.contains(p)){
   population.remove(p);
  }
 }

 /**
  *
  */
 public void bindPersonByAffinity(Person from, Person to,
 int affect, int trust, int admiration)
 throws NotInPopulationException {
  if (!population.contains(from)){
   throw (new NotInPopulationException()) ;
  }
  if (!population.contains(to)){
   throw (new NotInPopulationException()) ;
  }
  Affinity f = new Affinity(from, to, "" + affect, "" + trust, "" +admiration);
 }

 /**
  * round 2
  */
  public boolean isInNeighbourhood(Person source, Person target,
 String propertyName, String value){
  }

 // round 3

 /**
  *
  * @param args
  */
```

```
  public static void main(String[] args) {
  }

 /**
  *
  */
  public String toString(){
  }
}
```

- carcasse de l'interface Qualifiant

Il s'agit d'une classe abstraite, en fait

```
public abstract class Qualifier {

 protected Person from;
 protected Person to;

 /**
  *
  * @param f
  * @param t
  */
 public Qualifier(Person f, Person t){
  from = f;
  to = t;
 }

 /**
  *
  * @return
  */
 public Person getSource(){
  return from;
 }

 /**
  *
  * @return
  */
 public Person getTarget(){
  return to;
 }

 /**
  *
  * @param name
  * @return
  */
 public abstract String getQualifierByName(String name);

 /**
  *
  * @param name
  * @param value
  */
 public abstract void setQualifier(String name, String value);
}
```

- carcasse d'une classe Affinité implémentant l'interface précédente

```
public class Affinity extends Qualifier {

 protected int affect;
 protected int admiration;
```

```
 protected int trust;

 /**
  *
  * @param f
  * @param t
  */
 public Affinity(Person f, Person t) {
  super(f, t);
  // TODO Auto-generated constructor stub
 }

 /**
  *
  * @param f
  * @param t
  * @param aff
  * @param trst
  * @param adm
  */
 public Affinity(Person f, Person t, String aff, String trst, String adm) {
  super(f, t);
  affect = Integer.parseInt(aff);
  admiration = Integer.parseInt(adm);
  trust = Integer.parseInt(trst);
  // TODO Auto-generated constructor stub
 }

 @Override
 public String getQualifierByName(String name) {
  // TODO Auto-generated method stub
  if (name.equals("affect")){
   return "" + affect;
  }
  else if (name.equals("admiration")){
   return "" + admiration;
  }
  else if (name.equals("trust")){
   return "" + trust;
  }
  return null;
 }

 @Override
 public void setQualifier(String name, String value) {
  // TODO Auto-generated method stub
  if (name.equals("affect")){
   affect = Integer.parseInt(value);
  }
  else if (name.equals("admiration")){
   admiration = Integer.parseInt(value);
  }
  else if (name.equals("trust")){
   trust = Integer.parseInt(value);
  }
 }


}
```

Programmation de méthodes (4 points)

Ecrivez une méthode qui à partir de deux personnes données du réseau, détermine s'il existe un chemin qui passe de la première à la deuxième par un "chemin de confiance". (On appelera chemin de confiance un chemin dans le réseau où la confiance ne descend pas au dessous de 7 inclus).

Deux méthodes sont très fortement suggérées dans la classe Person, pour connaître les pairs. Pour réaliser cela, il faut explorer le réseau et donc pouvoir accéder à ce réseau. Comme nous sommes dans une personne, il faut que chaque personne connaisse le réseau auquel elle appartient :

```
public class Person{ ...

+++

  protected SocialNetwork network;


   // Constructors
   public Person(SocialNetwork parent){
    network = parent;
   }

   public Person(String fn, String ln, String nick,
 int age, char g, SocialNetwork parent){
    network = parent;
    firstname = fn;
    lastname = ln;
    nickname = nick;
    this.age = age;
    gender = g;
   }

+++
```

```
 // round 2

 public Person[] getTargets(){
 /*
  * using generic type
 Vector<Person> v = new Vector<Person>();

  for(Qualifier q : network.relationships){
   if (q.getSource() == this){
    v.add(q.getTarget());
   }
  }
  Person[] result = v.toArray(new Person[0]);
  return result;
    */


 /*
  * using raw Vector
  */
  Vector v = new Vector();

  for(Qualifier q : network.relationships){
   if (q.getSource() == this){
    v.add(q.getTarget());
   }
  }
  Object[] result = v.toArray(new Person[0]);
  return (Person[]) result;
 }

 /**
  *
  * @param name
  * @param threshold
  * @return
  */
 public Person[] getPeersByProperty(String name, String threshold){
  Vector v = new Vector();

  for(Qualifier q : network.relationships){
   Affinity a = (Affinity) q;
   int intValue = Integer.parseInt(a.getQualifierByName(name));
   int intThreshold = Integer.parseInt(threshold);
   if ((q.getSource() == this) && (intValue > intThreshold)){
    v.add(q.getTarget());
```

```
   }
  }
  if (v.isEmpty()){
   return null;
  }
  Object[] result = v.toArray(new Person[0]);
  return (Person[]) result;

 }
```

La première ne servant qu'à comprendre comment réaliser la deuxième. La deuxième est très intéressante puisqu'elle renvoie la collection des pairs qui accomplissent la contrainte sur une des propriétés. Il reste à implanter la méthode générale dans la classe SocialNetwork.

```
public class SocialNetwork{ ...

 /**
  * round 2
  */
 public boolean isInNeighbourhood(Person source, Person target,
 String propertyName, String value){

  Person[] myPeers;

  myPeers = source.getPeersByProperty(propertyName, value);
  if (myPeers != null){
   for (Person p : myPeers){
    if (isInNeighbourhood(p, target, propertyName, value)){
     return true;
    }
   }
  }
  return false;

 }
```

ATTENTION, cette implémentation n'est pas protégée contre les boucles : si le réseau d'affinités boucle, on trouvera toujours des Peers puisqu'on ne vérifie pas qu'on a déjà vu une personne. Cette implémentation est donc TRES INCOMPLETE d'un strict point de vue algorithmique. La solution qu'il faudrait mettre en place passerait de proche en proche une référence sur une collection accumulative qui "note" les personnes par lesquelles on est déjà passé et les élimine de la recherche.

Programmation d'application (6 points)

1. Ecrivez les constructeurs des classes ci-dessus et
2. un programme principal qui construit un jeu de test et imprime la construction.

Les constructeurs ayant déjà été donnés, on ne donne ici que les fonctions toString() et la main :

```
public class Person{


 ...


 //round 3
  public String toString(){
   String str = "";

   str += "Person[\n";
   str += "\tfirstname : " + firstname;
   str += "\tlastname : " + lastname;
   str += "\tnickname : " + nickname;
   str += "\tgender : " + gender;
   str += "]\n";

   return str;
  }
}
```

```
public class SocialNetwork{

 ...

 // round 3

 /**
  *
  * @param args
  */
 public static void main(String[] args) {
  // TODO Auto-generated method stub

  System.out.println("Hello\n");

  SocialNetwork sn = new SocialNetwork();
  Person p = new Person("Rachid", "Chelouah", "rc", 44, 'M', sn);
  sn.addPerson(p);
  p = new Person("Hervé", "de Milleville", "hdm", 52, 'M', sn);
  sn.addPerson(p);


  System.out.println(sn.toString());

 }

 /**
  *
  */
 public String toString(){
  String str = "";

  str += "Network[\n";
  str += "\tPopulation[\n";
  for (Person p : population){
   str += "\t" + p.toString();
  }

  str += "\t\t]\n";
  str += "\t]\n";

  return str;
 }
}
```

dont la sortie produit :

```
Hello


Network[
 Population[
 Person[
 firstname : Rachid lastname : Chelouah nickname : rc gender : M]
 Person[
 firstname : Hervé lastname : de Milleville nickname : hdm gender : M]
  ]
 ]
```

Barème :

- 1 point par constructeur
- 1 point pour la classe principale portant le main().