Université Paris Diderot **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques **Année 2015-2016**

TP n° 3

Héritage

Notes sur l'héritage :
Le but du TP est d'apprendre à se servir de l'héritage en Java. L'héritage permet, lorsque l'on définit une classe, de réutiliser une partie du code et des attributs de la classe parente à l'aide du mot clef **super**. Par exemple, si l'on souhaite définir des classes Vehicule et Voiture, on pourra écrire :

```
1  public class Vehicule {
2      private int nombreDeRoues;
3      public Vehicule(int n) {
4          this.nombreDeRoues = n;
5      }
6      public String toString() {
7          return "Ce vehicule a "
8              + this.nombreDeRoues + " roues.";
9      }
10 }
```

```
1  public class Voiture extends Vehicule {
2      private String couleur;
3      public Voiture(String couleur) {
4          super(4);   // Appelle le constructeur parent
5          this.couleur = couleur;
6      }
7      public String toString() {
8          String s = super.toString();    // Appelle la méthode
               parente
9          return s + "\n" + "Et c'est une voiture " + this.
               couleur + " !";
10     }
11     public static void main(String[] args) { // Permet de
           tester le code
12         Vehicule v = new Voiture("rouge");
13         System.out.println(v.toString());
14     }
15 }
```

Rappel (opérateur instanceof) :
L'expression x **instanceof** UneClasse vaut **true** si x est un objet de la classe UneClasse ou d'une classe héritière.

Exercice 1 Une situation simplifiée
On va commencer par une implémentation limitée de l'énoncé du TD. On se contente d'une classe Personne et de deux classe Noblesse et Roturiers héritant de la classe Personne. On ne fait pas de distinction entre l'argent et les produits

agricoles. Chaque noble ou roturier consomme une quantité fixe de ressources par an (mais le noble en dépense plus que les roturiers).
Implémentez une classe Test qui crée un noble dont dépend trois roturiers et qui les fait évoluer pendant 10 ans.

Exercice 2 Implémentation du TD
Créez une classe Societe correspondant à la modélisation que vous avez produite dans le dernier exercice du TD.

Exercice 3 Encryption symétrique

Notes sur le protocole d'encryption Un protocole d'encryption symétrique est la donnée d'une fonction de génération aléatoire de clés, d'une fonction d'encryption (qui prend en argument le message à encrypter et une clé, et qui retourne un message), et le protocole de décryption, qui prend en argument un message chiffré et la clé, et qui retourne le message déchiffré, ou une erreur si le message d'entrée ne correspond pas à un message chiffré avec la clé en question. On représente ici les messages par des entiers, et on notera longueur d'un message m le plus petit entier N tel que $m < 2^N$.

Notes sur le OR exclusif :
Java a un opérateur de OU Exclusif (xor) bit par bit sur les entiers dénoté : ^ .
Plus précisément, on peut considérer la classe suivante :

```
1  class BitwiseXOR {
2    public static void main(String args[]){
3
4    int num1 = 42;
5    int num2 = 15;
6
7    System.out.println("XOR Result =" +(num1 ^ num2));
8
9    }
10 }
```

En l'exécutant on obtient :

```
XOR Result = 37
```

En effet, 42 s'écrit en binaire 00101010, 15 s'écrit en binaire 00001111, et 37 s'écrit en binaire 00100101.

1. Écrire une interface EncryptionProtocol qui correspond à l'interface d'un protocole d'encryption. (On ajoutera une méthode **int** length_max(), qui est destinée à renvoyer la longueur maximale des messages que l'on veut chiffrer.)
2. Écrire une classe EncryptionProtocolId, qui réalise la classe EncryptionProtocol, et correspond à un protocole sans chiffrements : les deux fonction d'encryption et de décryption correspondent à l'identité.

3. Écrire une classe EncryptionProtocolOTP qui réalise la classe EncryptionProtocol qui correspond au protocole One-Time-Pad : la génération de clé est aléatoire, et l'encryption et la décryption correspondent au xor du message et de la clé

On veut maintenant représenter l'envoi d'un message entre un joueur A et un joueur B. La transmission des messages est assuré par une interface Reseau. Un réseau a une méthode communicate(Joueur,Joueur,**int**), qui correspond à envoyer un message d'un joueur vers un autre.

1. Écrire une classe Joueur munie de méthodes send et receive et ayant un attribut statique reseau, de type Reseau, et une méthode de classe **void** initialise (Reseau r), qui initialise la valeur de reseau à r. Ajoutez un attribut List<Integer> transcript, uttilisé pour stocker tous les messages reçus par le joueur depuis sa création.
   Ecrire également une interface Reseau qui a une méthode **void** communicate (Joueur,Joueur,**int**) qui correspond à envoyer un message d'un joueur vers un autre.
   Puisque toutes les communications se font par le réseau, la méthode sent de Joueur doit appeler la méthode communicate du réseau. La méthode **void** receive (**int** message) doit juste stocker lemessage dans transcript.
2. Écrire une classe JoueurEncrypted qui hérite de la classe Joueur et a comme attribut de classe un protocole d'encryption encr, et comme méthode de classe **void** initialise (EncryptionProtcole) qui initialise le protocole d'encyption. Cette classe vise à regrouper tous les joueurs qui connaissent le protocole d'encryption (et en particulier des joueurs malveillants).
3. Écrire une classe AliceAndBob qui hérite de la classe JoueurEncrypted et qui a une clé secrète comme attribut de classe, et une méthode **void** initialise () qui initialise la clé secrète en uttilisant la méthode de génération de clé de son attribut (hérité de JoueurEncrypted) encr. Cette classe vise à contenir les joueurs qui veulent communiquer secrètement entre eux grace à leur clé secrète.
4. Écrire une classe Test dans laquelle deux joueurs partageant la même clé secrète s'échangent un message.

Maintenant, on veut tenir compte du fait que le réseau n'est pas fiable. En particulier, il peut avantager plus ou moins un adversaire malveillant.

1. Écrire une classe ReseauHonnete, qui réalise l'interface Reseau, et qui correspond à un réseau qui transmet les messages sans les divulguer ni les modifier. On doit donc implémenter la méthode **void** communicate(Joueur j1, Joueur j2, **int** message) telle que la seule chose qu'elle fasse soit d'appeler la méthode receive du joueur j2.
2. Écrire une classe ReseauAdversairePassif implémentant l'interface Reseau, qui a comme attribut un joueur adversaire, et qui, lorsqu'il doit transmettre un message d'un joueur vers un autre, le communique également à l'adversaire
3. Écrire une classe ReseauAdversaireActif implémentant l'interface Reseau, qui laisse l'adversaire modifier les messages qui sont transmis.

Avec quelles combinaisons de protocoles d'encryption et de modèles de réseau peut-on considérer le protocole précédent comme sécurisé ? Imaginez un protocole qui soit sécurisé avec un adversaire passif mais pas un adversaire actif (et implémentez-le).