

Professeur : Hamid Mcheick

Groupe : Baptiste Buron, Matthieu Crouzet

Session : Hiver2017

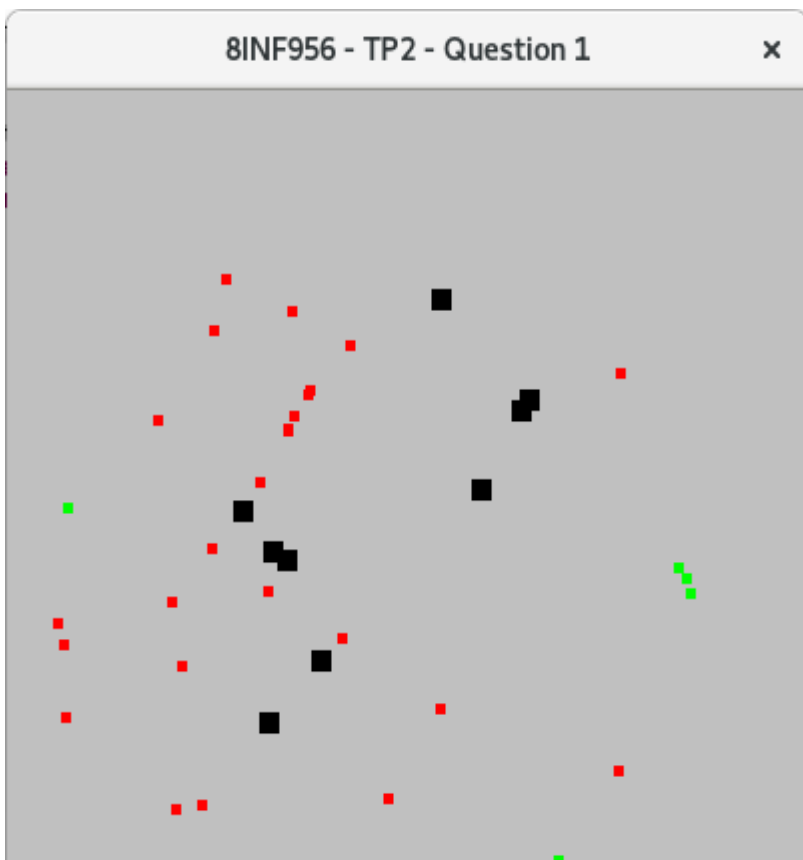
Date de distribution : 6 février 2017

Pondération : 10 points

Date de remise : 6 mars 2017

## Question 1

Pour ce travail, nous avons réalisé l'interface suivante :



- Un carré noir représente un pigeon
- Un petit carré vert représente de la nourriture fraîche.
- Un petit carré rouge représente de la nourriture avariée.

Le click gauche fait apparaître de la nourriture. Le click droit fait apparaître un pigeon.

La nourriture disparaît dès qu'un pigeon la touche. Chaque nourriture devient avariée au bout de dix secondes.

Si les pigeons ne trouvent plus de quoi se nourrir, aléatoirement ils peuvent être effrayés et se dispersent dans plusieurs sens.

Pour ce projet, nous avons établi la structure du code en quatre packages :

- *main* : contient le programme d'exécution
- *entities* : contient toutes les données relatives à l'environnement et au back-end de l'application (Food, Pigeon, Noise, World)
- *ui* : Représente la vue de l'application
- *utils* : Permet de gérer le mouvement des objets de l'application

### Héritage :

Nous utilisons l'héritage au sein des packages *entities* et *ui*. Par exemple, dans le package *ui*, les classes *Food* et *Pigeon* héritent de *Entity*.

### Multi-threading :

Concernant la programmation concurrente, nous avons implémentés l'interface *Runnable* afin d'implémenter les méthodes *run()* des classes *Pigeon*, *Food* et *World*, lancées simultanément au sein du programme. Ainsi nous avons plusieurs objets simultanément en interaction au sein du programme.

### Exceptions :

Nous gérons les exceptions au moyen de *try/catch*, par exemple dans le cas où les pigeons n'ont plus de nourriture et attendent :

```
private void sleep(){
    //System.out.println("Pigeon" + identifier + " sleeps during 2 seconds");
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

## Question 2

1) Voici trois exemples concrets de types de cohésion dans la programmation orientée objet :

### Cohésion entre les méthodes :

On obtient ce type de cohésion lorsque les méthodes de la classe coopèrent pendant l'exécution d'une fonctionnalité propre. Par exemple ; on peut avoir une cohésion procédurale où les méthodes agissent sur des activités potentiellement différentes, mais où elles peuvent faire transiter les données d'un endroit vers un autre. Par exemple on peut avoir une application de lecture média où les méthodes seraient focalisées et coopéreraient sur :

- l'ouverture du fichier
- la décompression des données
- le décodage des données
- la reproduction des données

### Cohésion entre les classes :

On dit que les classes sont en forte cohésion lorsque les attributs sont en relations avec les méthodes. Par exemple : On peut avoir une application avec une cohésion communicationnelle où les activités utilisent les mêmes paramètres d'entrées ou de sorties ?

Exemple : l'authentification via un login et un mot de passe :

```

public class UserManager {

    private static UserManager instance;

    private UserManager() {
    }

    public void manageUser(final String login, final String password)
        throws ClassNotFoundException, SQLException {
        ///
    }

    public static UserManager getInstance(){
        ///
    }
}

```

```

public class UserCredentialChecker {

    private static UserCredentialChecker instance;

    public boolean check(final String login, final String password)
    {
        ///
    }

    public static UserCredentialChecker getInstance(){
        ///
    }
}

```

```

public class UserDao {

    private static UserDao instance;

    public User retrieve(final String login, final String password)
    {
        ///
    }

    public static UserDao getInstance(){
        ///
    }
}

```

```

public class GuiMessageFormater {

    private static GuiMessageFormater instance;

    public String formatUIMessage(final User user){
        ///
    }

    public static GuiMessageFormater getInstance(){
        ///
    }
}

```

La classe UserManager laisse les spécificité aux autres classes spécialisées dans leur tâches.

### Cohésion au niveau de l'héritage :

Ce type de cohésion est semblable à la cohésion de classe mais on rajoute le concept des éléments hérités. Par exemple : dans le cas de la cohésion fonctionnelle ou on veut calculer les salaires de tous les employés selon leurs fonctions et leurs heures de travail. On pourrait visualiser des niveaux de hiérarchie : Personne -> Employé -> Poste.

## 2) LCOM : Mesure le manque de cohésion de la classe.

Pour calculer cette métrique, on compte le nombre de paires de méthodes qui n'accèdent pas aux mêmes attributs, puis on les soustrait au nombre de paires de méthodes qui accèdent aux mêmes attributs. Plus la valeur est élevée, plus le manque de cohésion est élevé.

```
public class CashRegister
{
    public static final double QUARTER_VALUE = 0.25;
    public static final double DIME_VALUE = 0.1;
    public static final double NICKEL_VALUE = 0.05;
    public static final double PENNY_VALUE = 0.01;
    private double purchase;
    private double payment;

    public CashRegister () {
        purchase = 0;
        payment = 0;
    }
    public void recordPurchase(double amount) {
        purchase = purchase + amount;
    }
    public void receivePayment(int dollars, int quarters, int dimes, int nickels, int pennies)
    {
        payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE + nickels *
NICKEL_VALUE + pennies * PENNY_VALUE;
    }
    public double giveChange() {
        double change = payment - purchase;
        purchase = 0;
        payment = 0;
        return change;
    }
}
```

Q = Paires de méthode qui ont au moins un attribut en commun.

P = Paires de méthodes qui n'ont pas d'attribut en commun.

(CashRegister(), recordPurchase()) attributs en commun : purchase

(CashRegister(), receivePayment()) attributs en commun : payment, QUARTER\_VALUE...

(CashRegister(), giveChange()) attributs en commun : purchase, payment

(recordPurchase(), receive Payment()) : pas d'attribut en commun

(recordPurchase(), giveChange()) attributs en commun : purchase

(receivePayment(), giveChange()) attributs en commun : payment

P= 1, Q = 5 : LCOM = |P| - |Q| = 0. On en déduit que le manque de cohésion est nul.

3) Prenons l'exemple suivant : imaginons deux classe où leur métrique LCOM est égale à 0. Il se peut que l'une des deux classe ait plus de connexions entre les attributs et les méthodes que l'autre. Cette métrique n'est donc pas forcément suffisante pour déduire de la qualité de la cohésion entre les méthodes et les attributs au sein d'une classe. Cependant, elle peut servir de base pour se donner une idée.

**4) Selon la LCOM de Henderson-Sellers :** On évalue la cohésion de la classe en fonction de la proportion d'appel aux attributs par les méthodes.

$LCOM = (\text{Nombre d'attributs} - \text{Somme des attributs appelé par les méthodes} / \text{nombre de méthodes}) / \text{nombre de méthodes} - 1$

Dans le cas de la classe CashRegister : Quatre méthodes, Six attributs

QUARTER\_VALUE est appelée par receivePayment() ;

DIME\_VALUE est appelée par receivePayment() ;

NICKEL\_VALUE est appelée par receivePayment() ;

PENNY\_VALUE est appelée par receivePayment() ;

purchase est appelée par CashRegister(), recordPurchase() et giveChange() ;

payment est appelée par CashRegister(), receivePayment() et giveChange() ;

Somme des attributs appelé par les méthodes =  $1+1+1+1+3+3 = 10$ .

$LCOM = (6 - 10/4)/3 = 1,16$

Cette métrique indique un très grand manque de cohésion, on en conclut que la classe CashRegister n'est pas cohésive.

#### 5) Proposition pour rendre cette classe cohésive :

```
public class CashRegister
{
    private double purchase;
    private double payment;

    public CashRegister () {
        purchase = 0;
        payment = 0;
    }
    public void recordPurchase(double amount) {
        purchase = purchase + amount;
    }
    public void receivePayment(int dollars, int quarters, int dimes, int nickels, int pennies)
    {
        payment = dollars + quarters * 0.25 + dimes * 0.1 + nickels * 0.05 + pennies * 0.01;
    }
    public double giveChange() {
        double change = payment - purchase;
        purchase = 0;
        payment = 0;
        return change;
    }
}
```

Nous n'avons plus que deux attributs pour quatre méthodes :

Somme des attributs appelé par les méthodes =  $3+3 = 6$ .

$LCOM = (2 - 6/4)/3 = 0,16$ . Cette classe est devenue cohésive.