
Interrogation d'informatique

PCC-SCAN-ASINSA 2^{ème} année - Janvier 2020



Durée totale : 2h

Documents autorisés : Toutes notes personnelles ou du cours dans une limite de 10 feuilles recto-verso

Attention : les téléphones portables sont interdits.

- Le barème est indicatif et le sujet est sur 6 pages.
- Les exercices sont indépendants.
- Un programme **mal indenté, mal commenté** ou avec de **mauvais choix de noms de variables** sera **sanctionné** (*jusqu'à -1 point*).

Barème indicatif :

- Partie 1 : 7.5 points
- Partie 2 : 12.5 points

Dernières précisions importantes à lire attentivement :

- les 2 parties sont indépendantes. Vous pouvez les traiter dans l'ordre que vous voulez.
- pour chaque question, vous devrez utiliser (dans la mesure du possible) les méthodes présentées dans les questions précédentes, même si vous ne les avez pas codées ;
- l'efficacité des algorithmes proposés fera l'objet d'une attention particulière ;
- le code java donné dans vos réponses devra être le plus concis possible et donc exploiter le plus possible les structures d'héritage ;
- la propreté de votre copie sera pris en compte dans la notation.

1 Modélisons les personnels de l'INSA (7.5 pts)

En tirant profit de la Programmation Orientée Objet (POO), nous souhaitons modéliser les différents membres de la famille INSA (que nous appellerons les *INSAliens*), en particulier 3 sous-catégories : les étudiants, les enseignants et les agents en CDI (contrat à durée indéterminée). Tous les *INSAliens* sont caractérisés par un nom et un prénom. Parmi les *INSAliens*, nous distinguons :

- Les *enseignants* caractérisés par leur indice (un entier qui définit leur grille de salaire) et la matière qu'ils enseignent. Par défaut cet indice vaut 463 pour un enseignant en début de carrière.
- Les *agents en CDI* caractérisés par leur salaire brut.
- Les *étudiants* caractérisés par un numéro d'étudiant (une chaîne de caractères) et un département (la classe `Département` est définie ci-dessous)

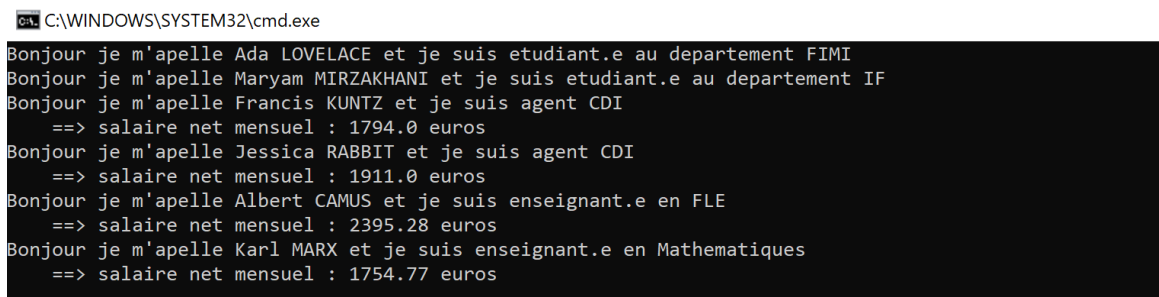
Les enseignants et les agents en CDI sont des *employés* de l'INSA ; ils touchent un salaire mensuel net calculé comme explicité ci-dessous (les chiffres sont fantaisistes) :

- Pour les enseignants : salaire net = indice * 3.79
- Pour les agents en CDI : salaire net = salaire brut * 0.78

Le programme principal ci-dessous illustre un exemple d'utilisation de ces différents objets. Proposez une hiérarchie de classes qui permette à la classe `TestInsa` de fonctionner correctement et d'obtenir le résultat attendu illustré dans la figure 1. Vous donnerez l'implémentation complète des classes (attributs et méthodes). Attention vous prendrez soin de tirer profit au maximum des mécanismes de la POO en factorisant et en réutilisant au maximum le code. Aucune interface (au sens des interfaces Java) n'est nécessaire ici. Avant d'écrire le code, nous vous conseillons fortement de faire le diagramme représentant la hiérarchie des classes (par

exemple au format UML). Ce diagramme pourra figurer sur votre copie mais n'entrera pas en compte dans le barème de la question.

```
1 public class TestInsa {
2     public static void main(String[] args){
3         INSAlien[] INSA = new INSAlien[6];
4
5         INSA[0] = new Etudiant("Ada","LOVELACE","4012456","FIMI");
6         INSA[1] = new Etudiant("Maryam","MIRZAKHANI","4011235","IF");
7         INSA[2] = new AgentCDI("Francis","KUNTZ",2300);
8         INSA[3] = new AgentCDI("Jessica","RABBIT",2450);
9         INSA[4] = new Enseignant("Albert","CAMUS",632,"FLE");
10        INSA[5] = new Enseignant("Karl","MARX","Mathématiques");
11        //si l'indice n'est pas précisé alors l'indice par défaut est utilisé: 463
12
13        for(int i=0; i<INSA.length; i++){
14            System.out.println(INSA[i]);
15            if (INSA[i] instanceof Employe) { //on vérifie si le type de INSA[i] est
16                //une classe fille de Employe
17                Employe s = (Employe)INSA[i];
18                System.out.println(" ==> salaire net mensuel :
19                    "+s.getSalaireNetMensuel()+" euros");
20            }
21        }
22    }
23 }
24
25 public class Departement {
26     private String nomDepart;
27
28     public Departement(String nom){
29         nomDepart=nom;
30     }
31
32     public String toString(){
33         return nomDepart;
34     }
35 }
```



```
C:\WINDOWS\SYSTEM32\cmd.exe
Bonjour je m'apelle Ada LOVELACE et je suis etudiant.e au departement FIMI
Bonjour je m'apelle Maryam MIRZAKHANI et je suis etudiant.e au departement IF
Bonjour je m'apelle Francis KUNTZ et je suis agent CDI
==> salaire net mensuel : 1794.0 euros
Bonjour je m'apelle Jessica RABBIT et je suis agent CDI
==> salaire net mensuel : 1911.0 euros
Bonjour je m'apelle Albert CAMUS et je suis enseignant.e en FLE
==> salaire net mensuel : 2395.28 euros
Bonjour je m'apelle Karl MARX et je suis enseignant.e en Mathematiques
==> salaire net mensuel : 1754.77 euros
```

FIGURE 1 – Illustration de l'exécution du programme TestInsa ci-dessus.

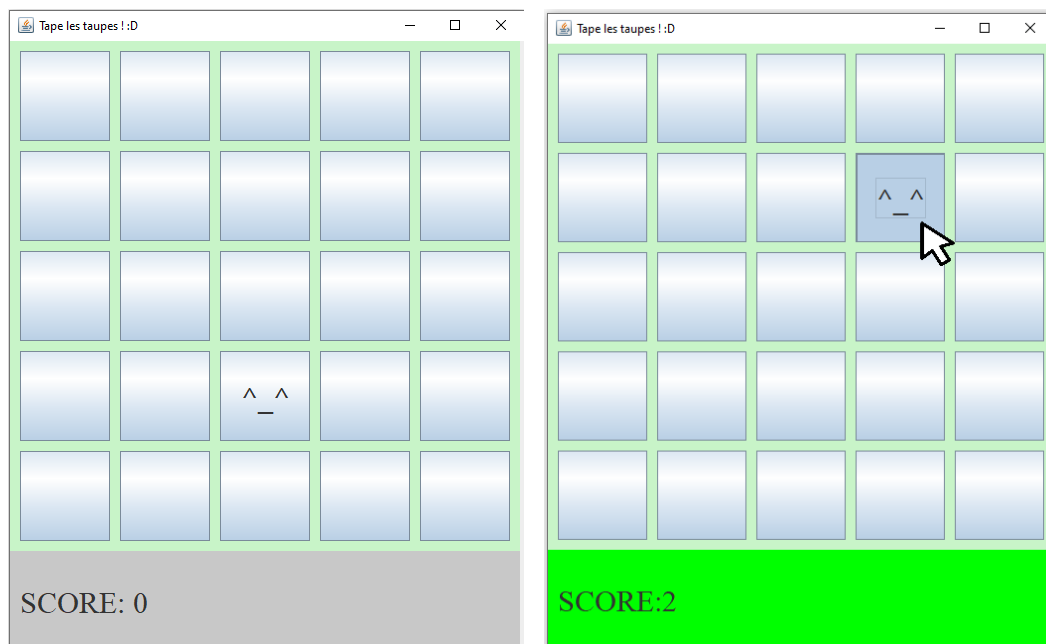
2 Tape taupe ! (12.5 pts)

Le tape-taupe (aussi appelé *jeu de la taupe* ou *Whac-A-Mole*) est un jeu d'adresse sur borne d'arcade. Il consiste à taper sur une taupe quand elle sort la tête d'un trou (voir la figure 2). Dans cet exercice, nous souhaitons réaliser un programme de type tape-taupe, tel qu'illustré dans la figure 3. Les "trous" sont représentés par 25 JButtons. Lors de la phase de jeu, seul un bouton, parmi les 25 présents, contiendra une taupe (représentée par la chaîne de caractères "^_^" affiché sur le bouton correspondant). Le joueur doit cliquer sur le bouton représentant la taupe. S'il le fait, le score est augmenté de 1 et la taupe est "déplacée" aléatoirement sur un autre bouton. Si le joueur clique sur un bouton sans taupe, le score est diminué de 1 et la taupe n'est pas "déplacée". Lorsque le score arrive à 10 points ou bien lorsqu'il est inférieur à 0, une seconde fenêtre apparaît qui affiche respectivement "Gagné" ou "Perdu".



(Source : Larry KING)

FIGURE 2 – Exemple de Tape taupe.



((a)) Au démarrage

((b)) Pendant le jeu

FIGURE 3 – Exemple de visuels attendus pour le jeu

(Q2.1) Mise en place des composants graphiques

La fenêtre principale de jeu est modélisée par 2 JPanel : *panneauDeBoutons* qui contient les 25 JButton, et *panneauMsg* qui contient le JLabel *msg* qui affiche le score. La classe *FenetreJeu* est détaillée en partie ci-dessous. La position de la taupe est modélisée par l'entier *taupe* qui définit l'indice du bouton correspondant. On admet que tous les import sont faits.

```
1 public class FenetreJeu extends JFrame implements ActionListener{
2     private JLabel msg;
3     private JPanel panneauDeBoutons;
4     private JButton[] lesBoutons;
5     private JPanel panneauMsg;
6     private int score;//Score du joueur
7     private int taupe;//Position de la taupe (entier entre 0 et 24).
8
9     public FenetreJeu(){
10        super("Tape les taupes ! :D");
11        setSize(530,660);
12        setLayout(null);
13        score = 0;
14
15        panneauDeBoutons = new JPanel();
16        panneauDeBoutons.setLayout(null);
17        panneauDeBoutons.setBounds(0,0,510,510);
18        panneauDeBoutons.setBackground(new Color(200,244,200));
19        add(panneauDeBoutons);
20
21        panneauMsg = new JPanel();
22        panneauMsg.setLayout(null);
23        panneauMsg.setBounds(0,510,510,100);
24        panneauMsg.setBackground(new Color(200,200,200));
25        add(panneauMsg);
26
27        //Initialisation et placement du JLabel msg
28        ...A COMPLETER (Q2.1.1)...
29
30        //Initialisation et placement des boutons
31        ...A COMPLETER (Q2.1.2)...
32
33        placeTaupe();//placement aléatoire et affichage de la taupe
34        setVisible(true);
35    }
36
37    public void actionPerformed(ActionEvent evt) {
38        for(int i=0;i<lesBoutons.length;i++){
39            if(evt.getSource()==lesBoutons[i]){
40                gererClic(i);
41                winOrLose();
42            }
43        }
44    }
45    ... A COMPLETER...
46 }
```

(Q2.1).1 Complétez le code du constructeur `FenetreJeu()` ci-dessus, afin d'initialiser et placer le `JLabel msg`. Le placement pourra être approximatif mais devra être correct.

(Q2.1).2 Complétez le code du constructeur `FenetreJeu()` ci-dessus, afin d'initialiser et placer les 25 `JButton` présents dans le tableau `lesBoutons`, qui représenteront les trous de taupe. Ces boutons devront envoyer des événements lorsqu'ils seront cliqués. Votre code doit pouvoir facilement s'adapter si le nombre de boutons change en fonction de la difficulté du jeu.

(Q2.2) Gestion du jeu

(Q2.2).1 Donnez l'entête et l'implémentation de la méthode `placeTaupe`. Cette méthode fait bouger la taupe aléatoirement et met à jour l'affichage sur les boutons. Rappel : la classe `JButton` peut utiliser la méthode `setText(String texte)`. On rappelle que l'instruction `Math.random()` renvoie un nombre aléatoire, de type double, dans l'intervalle $[0,1[$.

(Q2.2).2 Donnez l'entête et l'implémentation de la méthode `gererClic` appelée dans le `ActionPerformed` (voir le code source ci-dessus) et qui gère les fonctionnalités suivantes : si le bouton sur lequel l'utilisateur vient de cliquer contient la taupe alors la taupe doit être déplacée, le score doit être augmenté de 1 et le fond du `JPanel panneauMsg` devient vert. Sinon, la taupe reste en place, le score est diminué de 1 et le fond de `panneauMsg` devient rouge. Dans tous les cas, le nouveau score doit être affiché.

(Q2.2).3 Donnez l'entête et l'implémentation de la méthode `winOrLose` appelée dans le `ActionPerformed` (voir le code source ci-dessus) et qui vérifie si l'utilisateur a gagné (i.e. lorsque son score est égal à 10) ou perdu (i.e. lorsque son score est inférieur à 0). Si il a gagné ou perdu alors la taupe disparaît de l'affichage et une fenêtre de type `FenetreResultat`, dont le code et une illustration sont ci-dessous, est affichée (avec le message "Gagné" ou "Perdu" selon les cas).

```
1 public class FenetreResultat extends JFrame{
2
3     private boolean gagne;
4     public FenetreResultat(boolean aGagne){
5         gagne=aGagne;
6         this.setSize(300,150);
7         this.setLocation(700,200);
8         this.setVisible(true);
9     }
10
11     public void paint(Graphics g){
12         if(gagne){
13             g.setColor(Color.green);
14             g.fillRect(0,0,this.getWidth(),this.getHeight());
15             g.setColor(Color.black);
16             g.drawString("Gagne", 100, 110);
17         }
18         else{
19             g.setColor(Color.red);
20             g.fillRect(0,0,this.getWidth(),this.getHeight());
```

```
21     g.setColor(Color.black);
22     g.drawString("Perdu", 100, 110);
23 }
24
25 }
26
27 }
```



FIGURE 4 – Exemple d’une instance de la classe FenetreResultat

(Q2.3) Amélioration

Nous souhaitons maintenant améliorer le programme en faisant en sorte que la taupe change toute seule de position toutes les secondes (dès la construction de la fenêtre).

- (Q2.3).1 Quel composant faut-il utiliser pour cela? et quelles méthodes doivent être modifiées (les modifications ne sont pas demandées pour cette question).*
- (Q2.3).2 Donnez toutes les modifications à effectuer dans la classe FenetreJeu.*
- (Q2.3).3 On souhaite maintenant compter le temps (en secondes) mis par le joueur pour gagner (ou perdre). Ce temps sera affiché dans la FenetreResultat affichée après sa victoire (ou défaite) tel qu’illustré ci-dessous. Donnez toutes les modifications à effectuer dans les classes FenetreJeu et FenetreResultat pour mettre en place cette fonctionnalité.*

