Eléments de programmation par objets avec Java
Projet 2019 : "Créatures et avatars"

- Travail en binôme, temps total de travail attendu (environ 6 heures)
- Le projet est noté sur 20 points :
  - Avoir fait Partie 1 (2 points), Partie 2 (2 points), Partie 3 (2 points)
  - Partie 4 : Q15 + Jeu qui fonctionne (2 points), Q16 (1 point), Q17 (1 point), Q18 (2 points)
  - Partie 5 : Q20 (4 points)
  - Qualité du code : indentation, lisibilité, respect conventions... (2 points)
  - Qualité de la présentation orale (2 points ; note adaptée à chaque étudiant du binôme)
- Soutenance des projets pendant la **dernière séance de TME**. Déroulement :
  - 2 minutes maximum : Présentez une exécution de votre programme qui montre que votre programme fonctionne et qui mette en valeur ce que vous avez fait (jusqu'où êtes-vous allé ? quels items avez-vous rajouté ? quelles améliorations ou originalités avez-vous dévelopé ?...)
  - 8 minutes environ : Questions de l'enseignant pour vérifier : votre avancée dans le projet, que c'est bien vous qui avez fait les classes et que chaque étudiant du binôme a travaillé.
- Au fur et à mesure du projet, vous serez de moins en moins guidé pour la rédaction des classes. A vous de réfléchir pour ajouter des constructeurs, accesseurs, attributs... si nécessaires. Pensez à bien respecter les conventions d'écriture et les bonnes pratiques apprises en cours (attribut privé...), à tester dans un main les méthodes au fur et à mesure...
- Si vous ne savez pas répondre à une question, vous pouvez mettre une valeur par défaut qui vous permet de compiler et de pouvoir faire les autres questions. Essayez d'avoir un programme qui s'exécute pour la soutenance, même si vous n'avez pas pu répondre à chaque question.

Le projet consiste à réaliser un mini-jeu dans lequel des avatars (joueurs) se déplacent dans un monde peuplé de créatures et d'accessoires. Lorsqu'un avatar rencontre :

- un accessoire, il peut le ramasser
- une créature, il peut devenir ami avec la créature en lui offrant un accessoire

Les avatars ont 5 tours pour ramasser des accessoires et rencontrer des créatures. Puis une course est organisée. Le joueur gagnant est celui dont les créatures amies sont les plus rapides.

Partie 1 – Les accessoires

Il existe de nombreux accessoires, mais pour l'instant, on considère seulement les pommes et les sacs.

- Un sac est un accessoire. Le poids d'un sac est la somme des poids des accessoires qui sont dans le sac. Remarque : un sac peut contenir un sac.
- Une pomme est aussi un accessoire, mais elle a en plus la propriété d'être mangeable. On suppose que le poids d'une pomme de rayon r cm est $\frac{4}{3000}\pi r^3$ kg.

Q1 Au brouillon, faire un schéma UML pour modéliser les relations d'héritages Java entre les classes `Acc` (accessoire), `Avatar`, `Creature`, `Monde`, `Personnage`, `Pomme` et `Sac`.

Q2 Ecrire la classe `Acc` (accessoire) qui contient au moins les attributs et méthodes suivants :

- `cpt` : un compteur qui compte le nombre d'accessoires
- `numero` : le numéro de l'accessoire (1 pour le 1er, 2 pour le 2ième...)
- `categorie` : la catégorie de l'accessoire (sac, pomme,...)
- constructeur `Acc(String categorie)`
- méthode `double getPoids()` qui retourne le poids de l'accessoire. Aide : pouvez-vous définir le corps de cette méthode dans cette classe ?
- méthode `toString()` qui retourne (exemples) : `pomme No 1 0.26kg`, `sac No 2 1.80kg`

Aide 1 : pour afficher un réel avec seulement 2 chiffres après la virgule, utilisez :
`String s=String.format("chaine=%s entier=%d reel=%.2f", uneChaine, unEntier, unReel)`
Aide 2 : même si on ne peut pas définir le corps de la méthode `getPoids()` dans cette classe, on peut quand même utiliser cette méthode dans la classe `Acc` car le corps sera défini dans les classes filles

Q3 Ecrire la classe `Sac` qui contient au moins les attributs et méthodes :

- `tab` : un tableau d'accessoires (obligatoire : un seul attribut autorisé dans cette classe)
- constructeur `Sac(int n)` qui crée un sac pouvant contenir au maximum `n` accessoires
- constructeur sans paramètre qui crée un sac pouvant contenir entre 1 et 10 accessoires (aléatoirement)
- méthode `int size()` qui retourne le nombre maximal de places dans le sac
- méthode `ajouter(Acc a)` qui ajoute l'accessoire `a` à la première place libre, ou affiche le message `"Pas de place"` si aucune place n'est libre dans le sac
- méthode `Acc obtenir(int i)` qui retourne l'accessoire à la position i dans le sac si il existe, `null` sinon. L'accessoire retourné ne doit plus être dans le sac
- méthode `toString()` qui retourne (vous aurez besoin d'utiliser le `toString()` de `Pomme`) :
```
sac No 3 0,80kg contient 2 accessoires sur 7 places
     pomme No 1 0,26kg 3,9cm
     pomme No 2 0,54kg 5,0cm
```

Interface `Mangeable` Par la suite, certains éléments du monde pourront avoir la propriété d'être mangeable. Soit l'interface `Mangeable` suivante :
```
public interface Mangeable {
     public double getPoids();
}
```
Le fait qu'il existe déjà une méthode abstraite `getPoids()` dans `Acc` ne pose pas de problèmes. Il n'y aura au final qu'un seul corps de méthode dans l'objet.

Q4 Une pomme a la propriété d'être mangeable. Ecrire la classe `Pomme` qui contient au moins les attributs et méthodes suivants :

- `rayon` : le rayon (`double`) de la pomme (en cm)
- constructeur sans paramètre qui initialise aléatoirement le rayon de la pomme dans $[3, 7[$
- méthode `toString()` qui retourne : `pomme No 1 0,26kg 3,9cm`

Q5 Pour tester, dans le `main` d'une classe `Jeu`, ajoutez 2 pommes dans un sac, puis affichez le sac.

Partie 2 – Les personnages

Q6.1 Ecrire la classe `Personnage` qui contient au moins les attributs et méthodes suivants :

- `nom` : le nom du personnage
- `poids` : le poids du personnage
- constructeur `Personnage(String nom, double poids)`
- constructeur `Personnage(String nom)` qui initialise aléatoirement le poids dans $[30, 130[$
- la méthode suivante qui ajoute du poids au personnage :
`protected void addPoids(double p){if (p>0) {poids+=p;} }`
- une méthode `toString()` qui retourne par exemple : `Tulipo 77,3 kg`

Q6.2 Modifiez votre classe `Personnage` pour qu'on ne puisse pas créer d'instance de cette classe.
Q6.3 Les personnages de ce jeu ne veulent pas que tout le monde puissent connaître leur poids (sauf par `toString()`). Ajoutez à votre classe `Personnage` les instructions pour que seules les classes filles puissent connaître le poids du personnage, mais pas le modifier (sauf par `addPoids(double p)`), sachant que le poids d'un personnage peut varier.

Q7 On veut écrire une méthode `getNom()` qui retourne un nom de créature différent à chaque fois. Par exemple, si dans un tableau en attribut, on donne les noms `"Bob"`, `"Fil"` et `"Jaz"`, des appels successifs à cette méthode permettent d'obtenir les noms suivants : `"Bob1"`, `"Fil1"`, `"Jaz1"`, `"Bob2"`, `"Fil2"`, `"Jaz2"`, `"Bob3"`... Ecrire la classe outil `Noms` qui contient au moins :

- un attribut `tabNoms` de type tableau contenant des noms que vous choisissez.
- la méthode `getNom()`.

Aide : il vous faut 2 autres attributs : un pour parcourir le tableau et un autre pour compter le nombre de fois que le tableau a été parcouru.

Q8 Une créature est un personnage qui a un sac. Elle est capable de courir plus ou moins vite en fonction de son poids et du poids du sac. Ecrire la classe `Creature` qui contient au moins les attributs et méthodes suivants :

- `leSac` : un sac pouvant contenir entre 1 et 10 accessoires
- constructeur sans paramètre (aide : utiliser la méthode `getNom()` de la classe `Noms`)
- méthode qui ajoute un accessoire dans le sac de la créature si le poids du sac plus le poids de l'accessoire est inférieur à la moitié du poids de la créature, ou sinon affiche `"Trop lourd"`
- méthode `getVitesse()` qui retourne la vitesse à laquelle peut courir la créature. Soit p_c le poids de la créature et p_s le poids de son sac alors la vitesse de la créature est : $v = \frac{1}{4}p_c - p_s$, si $v < 0$ alors sa vitesse vaut 0.
- méthode `manger()`. La créature mange les accessoires mangeables du sac ce qui augmente son poids. Pour cela, on sort un par un les accessoires du sac. Si l'accessoire est mangeable alors on ajoute le poids de l'accessoire au poids de la créature, sinon on remet l'accessoire dans le sac. Remarque : pour simplifier si l'accessoire est un sac, la créature ne mange pas le contenu de ce sac
- méthode `manger(Mangeable m)` qui ajoute le poids du mangeable au poids de la créature
- méthode `courir()` qui *affiche* par exemple :

```
Bob1 51,0 kg court à vitesse 11,29 km/h avec sac No 4 1,46kg contient 2
accessoires sur 10 places
     pomme No 5 0,38kg 4,5cm
     pomme No 6 1,08kg 6,4cm
```

Q9 Pour tester, dans le `main` de la classe `Jeu`, créez une créature, faites-la courir, ajoutez 3 pommes dans son sac, faites à nouveau courir la créature, puis faites-lui manger ses pommes, enfin, faites encore courir la créature.

Q10 Un avatar est un personnage qui a une liste d'amis et une liste d'accessoires. Ecrire la classe `Avatar` qui contient au moins les attributs et méthodes suivants :

- `listeAmis` : une liste au sens `ArrayList` de créatures
- `listeAcc` : une liste au sens `ArrayList` d'accessoires
- constructeur qui prend en paramètre le nom et le poids de l'avatar
- méthode `toString()` qui retourne par exemple `Jake 79.5 kg 1 ami(s) 3 accessoire(s)`
- méthode `estAmi` qui retourne vrai si la créature en paramètre est un ami, faux sinon.
- méthode `devenirAmi` qui fait devenir ami l'avatar avec la créature en paramètre (s'il n'est pas déjà ami avec) et affiche un message du style "X devient ami avec Y".
- méthode `perdreAmi` (inspirez-vous de la méthode précédente)
  Aide : voir méthode `boolean remove(Object o)` de `ArrayList`
- méthode `rencontrer(Creature c)`. Lorsque l'avatar rencontre une créature, il lui offre le premier accessoire qu'il a dans sa liste d'accessoires et la créature ajoute cet accessoire dans son sac. Si la créature n'est pas amie avec lui et si l'accessoire offert a un poids supérieur à 0.5kg alors la créature devient amie avec lui. Si, par contre, il n'a pas d'accessoires à offrir à la créature alors si cette créature était une amie, il perd son amitié. Aide : voir méthode `E remove(int index)` de `ArrayList`
- méthode `double course()` qui fait manger, puis courir pendant 1 heure chaque créature l'une après l'autre, et qui retourne la distance totale parcourue

- méthode `Creature getCreaturePlusRapide()` qui retourne la créature qui court le plus vite
- méthode `compterAccMangeable()` qui retourne le nombre d'accessoires mangeables

Partie 3 – Le monde

Les personnages et les accessoires se trouvent dans un monde. Un monde a une forme carrée (voir figure un peu plus loin). Chaque élément (item) dans un monde a un nom et des coordonnées : une abscisse x et une ordonnée y.

Q11 Soit la classe `Item` ci-dessous. Modifier les classes `Acc` et `Personnage` pour qu'elles héritent de cette classe. Le nom de l'accessoire est sa catégorie.

```
public abstract class Item {
    private final String nom;
    private int x, y;
    public Item(String nom, int x, int y) { this.nom=nom; this.x=x;this.y=y; }
    public Item(String nom) { this(nom,-1,-1); }
    public int getX() { return x; }
    public int getY() { return y; }
    protected void setX(int x) { this.x=x; }
    protected void setY(int y) { this.y=y; }
    public String getNom() { return nom; }
    public String toString() { return nom+"("+x+","+y+")"; }
    public int distance(Item item) { // distance de Manhattan
        return Math.abs(x-item.x)+Math.abs(y-item.y);
    }
}
```

Q12 Ecrire la classe `Monde` qui contient au moins :

- `listeItems` : une liste au sens `ArrayList` d'items
- `taille` : la taille du monde
- méthode `int getPositionAlea()` qui retourne une position (un entier) aléatoire dans le monde (cette méthode sera utilisée pour obtenir des valeurs aussi bien pour les abscisses que pour les ordonnées)
- méthode `ajouterItem(Item item)` qui initialise aléatoirement l'abscisse et l'ordonnée de l'item et ajoute l'item dans le monde
- méthode `supprimerItem(Item item)` qui supprime l'item du monde et met ses coordonnées à (-1,-1)
- méthode `Item chercher(int x, int y)` qui retourne le premier item trouvé aux coordonnées (x,y) ou retourne `null` sinon
- méthode `ArrayList<Item> getVoisins(Item item)` qui retourne la liste des items qui sont à une distance ≤ 2 de l'item en paramètre (la liste de voisins ne doit pas contenir l'item en paramètre)
- méthode privée et statique `String getNomCourt(String nom)` qui retourne une chaîne d'exactement 4 caractères. Par exemple : `"x"` (1 caractère) devient `"␣␣x␣"` (4 caractères) ; `"xx"` devient `"␣xx␣"` ; `"sac"` devient`"sac␣"` ; `pomme"` devient `"pomm"`. Aide : utiliser les méthodes `int length()` et `String substring(int beginIndex, int endIndex)` de la classe `String`.
- méthode `afficher()` qui affiche le monde avec ses items.

  Voici ci-contre un exemple d'affichage pour un monde de taille 5. Principe : pour chaque case de coordonnées (x,y), rechercher un item à ces coordonnées. Si plusieurs items sont aux mêmes coordonnées, afficher seulement le premier item trouvé. Chaque case a une largeur d'affichage de 4 caractères.

```
   |  0 |  1 |  2 |  3 |  4 |
 0 |Fil1|    |    |    |pomm|
 1 |    |Jake|pomm|Jaz1|    |
 2 |pomm|    |sac |    |    |
 3 |Fil2|    |    |    |    |
 4 |    |Bob1|    |Bob3|pomm|
```

Q13 Pour tester, dans le `main`, créer un monde, y ajouter des créatures, des pommes et un avatar, puis afficher le monde.

Q14 Comment peut-on faire pour que les avatars puissent interagir avec le monde (par exemple, se déplacer, ramasser un accessoire...) ? Une solution possible est de mettre un attribut de type `Monde` dans la classe `Avatar`. Ainsi le monde connaît ses items, mais un avatar connaît aussi son monde. Modifiez la classe `Avatar` pour que son constructeur prenne un monde en paramètre et ajoute l'avatar au monde. Puis ajoutez à cette classe `Avatar` les méthodes suivantes :

- méthode `ramasser(Acc a)`. Ramasser un accessoire signifie le supprimer du monde, et le mettre dans la liste d'accessoires. Cette méthode affiche également un message du style "X ramasse A".
- méthode `rencontrerVoisins()`. Pour chaque voisin de l'avatar (voir méthode `getVoisins()` de `Monde`) : si le voisin est un accessoire, alors l'avatar ramasse l'accessoire ; si le voisin est une créature, alors l'avatar rencontre la créature ; si le voisin est un avatar, alors l'avatar salue (affiche une salutation) l'autre avatar. Remarque : l'avatar ne doit pas se saluer lui-même.
- méthode `seDeplacer()` qui déplace l'avatar dans le monde.
  Le programme demande à l'utilisateur de saisir dans le terminal une abscisse. Si l'abscisse n'est pas dans le monde, alors le programme demande à nouveau de saisir une abscisse jusqu'à ce que le nombre saisi soit correcte. Si l'abscisse est dans le monde, alors on fait de même avec l'ordonnée (voir exemple ci-contre). On suppose que l'utilisateur saisie toujours un nombre entier. Aide : utiliser la classe `Scanner` du package `java.util`, en particulier sa méthode `nextInt()`.

```
### Déplacement de Jake ###
Entrer une abscisse entre [0,4] :
 20
Entrer une abscisse entre [0,4] :
 3
Entrer une ordonnée entre [0,4] :
 2
Déplacement de Jake de (1,1) vers (3,2)
```

Partie 4 – Le jeu et ses améliorations

Q15 (2 points) Le jeu de base consiste à générer un monde avec des créatures et des accessoires, puis à y placer deux avatars (représentants 2 joueurs humains). Ensuite, on réalise 5 tours de jeu. Pendant chaque tour, chaque joueur se déplace avec stratégie afin d'obtenir des accessoires et des amis parmi les créatures. Puis à la fin des 5 tours, chaque avatar réalise une course avec ses créatures. Le gagnant est celui dont les créatures ont parcouru la plus grande distance. Programmez ce jeu. *Présentation orale :* vous devez montrer que vous avez un jeu qui fonctionne et qui respecte les consignes.

Q16 (1 point) Peut-on ajouter des accessoires mangeables et les utiliser à la place des pommes sans modifier les classes déjà programmées ? Si vous avez bien suivi les règles de POO, la réponse est oui. Pour le vérifier, proposez et programmez un nouvel accessoire mangeable qui possède au moins une variable d'instance. *Présentation orale :* vous devez montrer le code de ce nouvel accessoire mangeable, le respect des consignes et l'originalité seront prises en compte.

Q17 (1 point) Peut-on ajouter des classes filles de `Creature` et les utiliser sans modifier les classes déjà programmées ? La réponse est encore oui. Pour le vérifier, proposez une nouvelle créature qui a un comportement particulier pour courir plus vite (ou moins vite) en fonction des (éventuellement nouveaux) accessoires qu'elle possède ou pas, puis utilisez-la dans le main sans modifier les autres classes. *Présentation orale :* vous devrez montrer et expliquer le code de cette nouvelle créature, le respect des consignes et l'originalité seront prises en compte.

Q18 (2 points) Proposez une hiérarchie de classes avec au moins trois niveaux pour des items du monde qui ne sont ni des accessoires ni des personnages, mais qui sont capables d'interargir avec des personnages. Par exemple, un pommier producteur de pommes qui interagit avec les personnages (la classe `Pommier` serait la classe fille de la classe `Arbre` elle-même fille de la classe `Item`, donc il y a bien 3 niveaux). *Présentation orale :* vous devrez montrer et expliquer les codes correspondant à cette hiérarchie, le respect des consignes et l'originalité seront prises en compte.

Partie 5 – Graphisme

Dans cette partie, on souhaite afficher le monde dans une fenêtre graphique.

Pour pouvoir faire du graphisme, charger au début de chaque classe qui contient une partie graphique, les classes des packages `java.awt` et `javax.swing` ainsi :

```
import java.awt.*;
import javax.swing.*;
```

On suppose que chaque `Item` peut être dessiné dans une fenêtre graphique. Dans la classe `Item`, ajoutez la méthode suivante qui dessine dans la fenêtre graphique un rectangle bleu à la position (x,y) de l'item en considérant que chaque case a une largeur de `tailleCase` (voir modification de la classe `Monde` ci-après) :

```
public void dessiner(Graphics g, Monde m) {
   int tc=m.getTailleCase();
   g.setColor(new Color(0,0,255)); // couleur courante devient bleu
   g.fillRect(getX()*tc, getY()*tc, tc, tc); // carré plein
}
```

La classe `JPanel` du package `javax.swing` va nous permettre de représenter le monde dans un panneau (*panel*) affichable dans une fenêtre graphique.
Documentation : `https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html`

Faites une copie de votre classe `Monde` dans un autre fichier, pour avoir une sauvegarde de cette classe que l'on va modifier. Puis, modifiez la classe `Monde` ainsi :

- modifiez le début du fichier par :
  ```
  import java.awt.*;
  import javax.swing.*;
  public class Monde extends JPanel { ...
  ```
- modifiez le début du constructeur pour initialiser les dimensions du panneau :
  ```
  public Monde(int taille, int tailleCase) {
      setPreferredSize(new Dimension(taille*tailleCase,taille*tailleCase));
      this.tailleCase=tailleCase;
      this.taille=taille;
  }
  ```
  avec `tailleCase` le nombre de pixels d'une case dans le panneau (ajouter un attribut `tailleCase` et son accesseur).
- compilez votre classe `Monde` et ajoutez si besoin les instructions nécessaires pour corriger les quelques erreurs qui ont pu apparaître.
- ajoutez enfin la méthode `paintComponent(Graphics g)` suivante qui permet de dessiner chaque item dans le panneau.

  ```
  public void paintComponent(Graphics g) {
     super.paintComponent(g); // redessine le panneau
     for(Item itemVoisin : listeItems) {
        if ( itemVoisin != null ) {
           itemVoisin.dessiner(g,this);
        }
     }
  }
  ```

Le programme ci-après vous donne un exemple d'affichage d'un monde dans une fenêtre graphique. Copiez-collez-le dans un fichier : `TestGraphisme.java`.

```
import java.awt.*;
import javax.swing.*;
public class TestGraphisme {
    private static final int TAILLE_CASE=30;
    private static final int NB_CASES=20;

    public static void main(String [] args) throws InterruptedException {
        // Création fenêtre graphique et ses caractéristiques
        JFrame f=new JFrame();
        f.setLocationRelativeTo(null);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Création du monde (qui est un panneau)
        Monde m=new Monde(NB_CASES,TAILLE_CASE);
        f.setContentPane(m); // Ajout du monde à la fenêtre
        f.pack(); // Adaptation de la fenêtre au panneau
        f.setVisible(true);

        Avatar jake=new Avatar("Jake",79.5,m); // ajoute Jake dans le monde
        for(int i=0;i<10;i++) {
            Thread.sleep(1000); // Ralenti l'affichage
            jake.seDeplacer();
            m.repaint(); // Redessine le graphique
        }
    }
}
```

Q19 Maintenant que l'on a une fenêtre graphique fonctionnelle avec une animation. Redéfinir dans les classes descendantes d'`Item` la méthode `dessiner` pour différentier l'affichage (choisissez les formes et les couleurs comme vous voulez). Par exemple, dans la classe `Pomme`, on peut redéfinir cette méthode ainsi :

```
public void dessiner(Graphics g, Monde m) {
        int tc=m.getTailleCase();
        g.setColor(new Color(255,0,0)); // couleur courante devient rouge
        g.fillOval(getX()*tc, getY()*tc, tc, tc); // cercle plein
}
```

Documentations des classes `Color` (pour la couleur) et `Graphics` (pour dessiner des rectangles, ovales... pleins (`fill`) ou vides (`draw`)) du package Java `java.awt` :

`https://docs.oracle.com/javase/8/docs/api/java/awt/Color.html`
`https://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html`

Vous pouvez forcer la redéfinition pour chaque sorte d'items, en déclarant la méthode `dessiner` abstraite dans la classe `Item`.

Q20 (4 points) Ajoutez des instructions dans la classe `TestGraphisme`, pour créer un monde peuplé d'items, de créatures et d'un ou plusieurs avatars. Inventez vos propres règles du jeu et faites tourner le jeu pour que les avatars intéragissent avec le monde. Idée : faire un menu avec un scanner (voir la classe `java.util.Scanner`) pour demander au joueur qu'est-ce qu'il veut faire : se déplacer, rencontrer une créature, donner un accessoire à une créature... *Présentation orale :* montrez que la fenêtre graphique s'affiche, que l'animation fonctionne (1 point). L'originalité et la qualité du jeu proposé seront pris en compte (3 points)