NOM (majuscules) : _____

Prénom(s) : _____

Groupe : _____

Contrôle M2103-- Bases de la programmation orientée objet

Consignes :

Vous avez **2h** pour répondre aux questions ci-dessous.

Vous pouvez utiliser l'examen pour écrire vos solutions dans les espaces données sous chaque question. Si vous voulez utiliser un autre support pour écrire vos solutions, **veuillez indiquer cela clairement** dans votre examen et veuillez également mettre votre nom et prénom **sur toute fiche utilisée** pour écrire vos solutions.

Veuillez **lire le préambule** de l'examen attentivement avant de commencer.

Vous avez le droit **à une page de notes personnelles recto/verso** (en français ou en anglais), à la **documentation Java sur les collections** et au bouquin **Java Récapitulation**. Vous n'avez pas le droit **à rien d'autre** (y compris calculatrice, téléphone portable, les feuilles de TD/TP).

La note finale du module est calculée à partir de : votre note de TD (15%), votre note de TP (15%) et votre note d'examen (70%). Tout bonus que vous avez gagné pour la note de l'examen s'applique seulement à cette note-ci, pas aux notes de TP/TD. Inversement, tout bonus pour le TD/TP s'applique seulement à cette note-là.

Note TD :

Note TP :

Note examen : / 33 = /20

Préambule

Il y a questions dans cet examen. Elles ne sont pas complètement indépendantes, car elles partagent un seul contexte. Toutefois, à chaque question vous allez commencer sur un diagramme de classe mis à jour, pour lequel vous pouvez supposer que toute méthode et tout attribut mis dedans sont déjà bien écrits.

On ne vous demande pas dans ce module de pouvoir faire des diagrammes de classe. Toutefois, on attend bien que vous soyez capable de les lire, sans avoir besoin d'autres explications. Il est conseillé à chaque exercice de bien regarder les diagrammes de classe avant d'écrire du code (au cas où il y a des méthodes que vous pouvez utiliser pour votre solution).

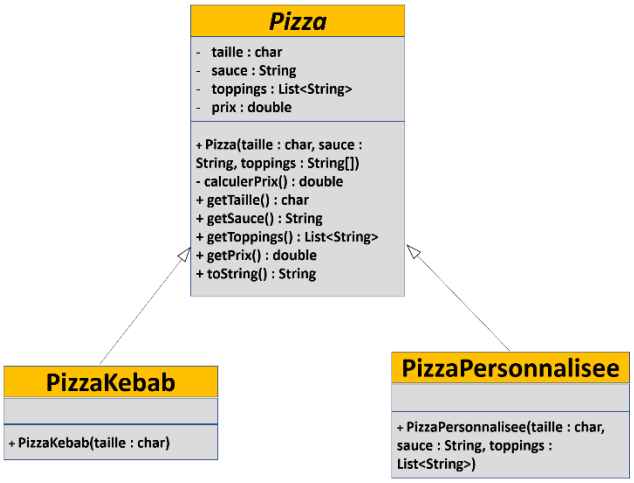
Pour les questions de type QCM : sauf autrement indiqué, toute question QCM dans cet examen peut avoir plusieurs réponses. Ne vous arrêtez pas à une seule réponse, sauf indiqué dans la question !

Contexte

En Français on dit « je joue du piano », mais aussi « je joue à MarioKart ». Cet examen utilise la similarité de langage entre ces deux phrases.

Nous aurons un environnement où des personnes auront la possibilité d'interagir avec des instruments musicaux ainsi qu'avec des jeux vidéo. Ils pourront acheter des instruments, des jeux ou encore de la nourriture. Ils pourront jouer à un instrument ou à un jeu vidéo seuls ou avec des amis.

Question I (10.5 points)



Comme tout le monde le sait, on ne peut pas vraiment passer une bonne soirée en jouant aux jeux ou aux instruments sans avoir de la nourriture, plus spécifiquement de la pizza. Une pizza aura une taille (M ou

L), un type de sauce (tomate, crème fraîche ou BBQ) et des toppings. Il y aura un prix de base pour la pâte et pour la sauce en fonction de la taille. Les toppings représentent des unités d'ingrédients qui sont mis sur la pizza : si on veut un supplément d'un ingrédient, celui-ci sera dupliqué dans la liste de toppings. Chaque topping coûtera un certain montant -- les premières toppings moins que celles d'après.

Nous voulons plus particulièrement réaliser du code qui nous aidera mettre en place le diagramme de classe ci-dessus.

1. **(1 point)** Une classe abstraite :

   ☐ Est une interface.
   ☐ Est une classe sans constructeur.
   ☐ Peut avoir plusieurs méthodes concrètes mais DOIT contenir au moins une méthode abstraite.
   ☐ N'est jamais instanciable.
   ☐ Est indiquée par du texte gras dans le diagramme de classe.
   ☐ Est indiquée par du texte italique dans le diagramme de classe.
   ☐ Ne peut jamais avoir une méthode concrète `String toString()`.

2. **(1 point)** Expliquez les notions de *getter* et *setter*.

3. La classe Pizza est abstraite et elle a quatre attributs : un attribut `taille` de type `char` (avec des valeurs possibles '`M`' et '`L`') ; un attribut `sauce` de type String (avec des valeurs "`tomate`" et "`creme fraiche`") ; un attribut `toppings` de type `List<String>` ; et un attribut `prix` de type `double`.
   Répondez aux questions suivantes :
   a. **(0.75 points)** Relisez le texte de l'exercice I. Est-ce qu'on pourrait changer le type de la variable toppings de `List<String>` à `Set<String>` ? Justifiez votre réponse.

b. **(1.25 points)** Le prix d'une pizza est calculé selon la formule suivante :

- Il y aura un prix de base de 6 euros pour une pizza M et 8 euros pour une pizza L.
- Chacune des premières trois toppings coûtera 0.5 euros.
- A partir de la troisième chaque topping coûte 0.25 euros.

Pour l'instant nous allons supposer que les valeurs mises par les usagers dans le reste du code seront correctes par rapport à tous les attributs en entrée : c'est-à-dire, ils vont toujours utiliser une taille soit de M, soit de L, ils vont seulement utiliser les valeurs admises pour les sauces, les toppings, etc.

Le prix d'une pizza est calculé selon la formule ci-dessus dans la méthode `void calculerPrix()`. Ecrivez cette méthode.

c. **(1 point)** Quelles affirmations parmi les affirmations suivantes sont correctes pour le constructeur d'une classe ?

☐ Chaque classe concrète DOIT contenir un constructeur.
☐ Une classe `ClasseExemple` pourrait contenir 2 constructeurs avec les signatures suivantes :

```
ClasseExemple(int parametre1, String parametre2, int parametre3)
ClasseExemple(int parametre3, String parametre2, int parametre1)
```

☐ Un constructeur DOIT instancier chaque attribut de la classe en question.
☐ Si une classe n'a aucun constructeur il est impossible d'instancier des objets de ce type.
☐ On appelle le constructeur de la classe `ClasseX` lorsqu'on écrit la ligne de code :

`System.out.println(<en objet de type ClasseX>`

- Si `Superclasse` est une superclasse de `Sousclasse` et `Superclasse` et `Sousclasse` ont des constructeurs avec la signature `Superclasse()` et `Sousclasse()` alors la ligne de code suivante appelle le constructeur de la classe `Superclasse` puisque l'objet créé a été défini comme étant de type Superclasse :

```
Superclasse objet = new Sousclasse() ;
```

d. **(0.5 points)** Le constructeur de la classe Pizza a la signature indiquée dans le diagramme de classe (attention au format de la variable toppings donnée en paramètre). Dans le constructeur on instancie les trois premières valeurs aux contenus des paramètres correspondants donnés en entrée. Puis, pour l'attribut prix on fait calculer le prix et on instancie l'attribut à la valeur calculée.
Ecrivez ce constructeur.

4. **(1 point)** La classe `Pizza` sert de superclasse à un nombre de sousclasses, tel qu'indiqué dans le diagramme de classe. Chacune de ces classes a seulement un constructeur, dont la signature est indiquée ci-dessus. Sachant qu'une pizza Kebab a une sauce de type crème fraîche et les ingrédients suivants : kebab, mozzarella, paprika, écrivez cette classe entièrement.

5. **(1 point)** Le code suivant est écrit dans la méthode principale (`main`) d'une classe `Examen2019`. Pour chaque ligne de code expliquez quelle(s) méthode est(sont) appelée(s), si le code compile ou non et quels problèmes elle peut entrainer.

```
Pizza royale = new Pizza('L', "tomate", new String[] {"jambon",
"champignons", "Emmental"});
Pizza kebab = new PizzaKebab('M');
```

6. **(2 points)** Le constructeur de la classe `PizzaPersonnalisee` prend en entrée trois paramètres, tel que décrit dans le diagramme de classe. Il instancie les trois premiers attributs de la pizza en utilisant les valeurs en entrée et finalement le prix de la pizza est calculé en fonction des entrées.

Dans une classe `Examen2019`, dans une méthode `main`, on écrit le code suivant.

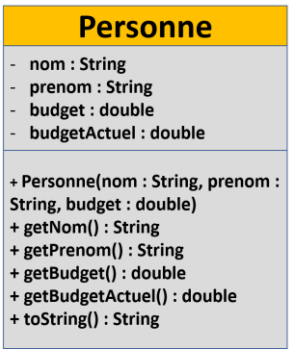
```
Pizza kebab1 = new PizzaKebab('M');
Pizza kebab2 = new PizzaPersonnalisee('M', "creme fraiche", new String[]
{"kebab", "mozzarella", "paprika"});
System.out.println(kebab1==kebab2);
System.out.println(kebab1.equals(kebab2));
```

Quelle est la sortie de ce code ? Justifiez votre réponse.

7. **(1 point)** Nous voulons faire en sorte que, lorsqu'on met une mauvaise valeur en paramètre pour le constructeur de la classe Pizza par rapport à sa taille ou à la sauce qu'on veut utiliser, une exception soit levée et l'utilisateur soit averti du problème. Notamment nous voulons utiliser un type d'exception appelée IllegalArgumentException.

   Ecrivez le code qu'il faut ajouter dans la classe Pizza pour que cette exception soit prise en compte, en expliquant dans quelle méthode et dans quelle partie de cette méthode il faut ajouter ce code.

Exercice II (6 points)



On peut acheter des pizzas. On peut acheter des instruments musicaux. On peut acheter des jeux vidéo. On peut jouer des instruments ou jouer aux jeux vidéo, mais on ne joue pas de la pizza. On peut accorder des instruments, mais pas des jeux. Nous allons grouper toutes ces différentes caractéristiques dans des fonctionnalités -- notamment en utilisant des interfaces.

Vous pouvez désormais supposer que le code des classes données à l'exercice 1 est bien écrit.

Nous allons, de plus, supposer l'existence d'une classe très basique appelée `Personne`, dont le diagramme de classe est donné ci-dessus.

1. **(1.5 points)** Nous aurons les interfaces suivantes : une interface `Achetable` avec une seule méthode `double getPrix()`, une interface `Accordable` avec une seule méthode `void accorder()` et une interface `Jouable` avec une méthode `void jouer(List<Personnes> participants)`. Ecrivez ces trois interfaces.

2. **(0.75 points)** Quel est le lien et quelles sont les différences entre `List<Personne>` et `ArrayList<Personne>` ?

3. **(1 point)** Lesquelles de lignes de code suivantes sont légitimes (en isolation, on considère à chaque fois que la ligne donnée est la seule ligne de code présente) ?

   - ☐ `List<Personne> personnes = new List<>() ;`
   - ☐ `List<Personne> personnes = new ArrayList<>() ;`
   - ☐ `ArrayList<Personne> personnes = new List<>() ;`
   - ☐ `List<String> strings ;`
   - ☐ `List<int> entiers ;`
   - ☐ `List<Personne> personnes ;`

4. **(0.75 point)** La classe `Pizza` doit fonctionner selon l'interface `Achetable`. Comment doit-on changer la classe `Pizza` pour que cela soit pris en compte ? (Indiquez où il faut changer le code et comment il faut le changer, mais ne recopiez surtout pas tout le code !)

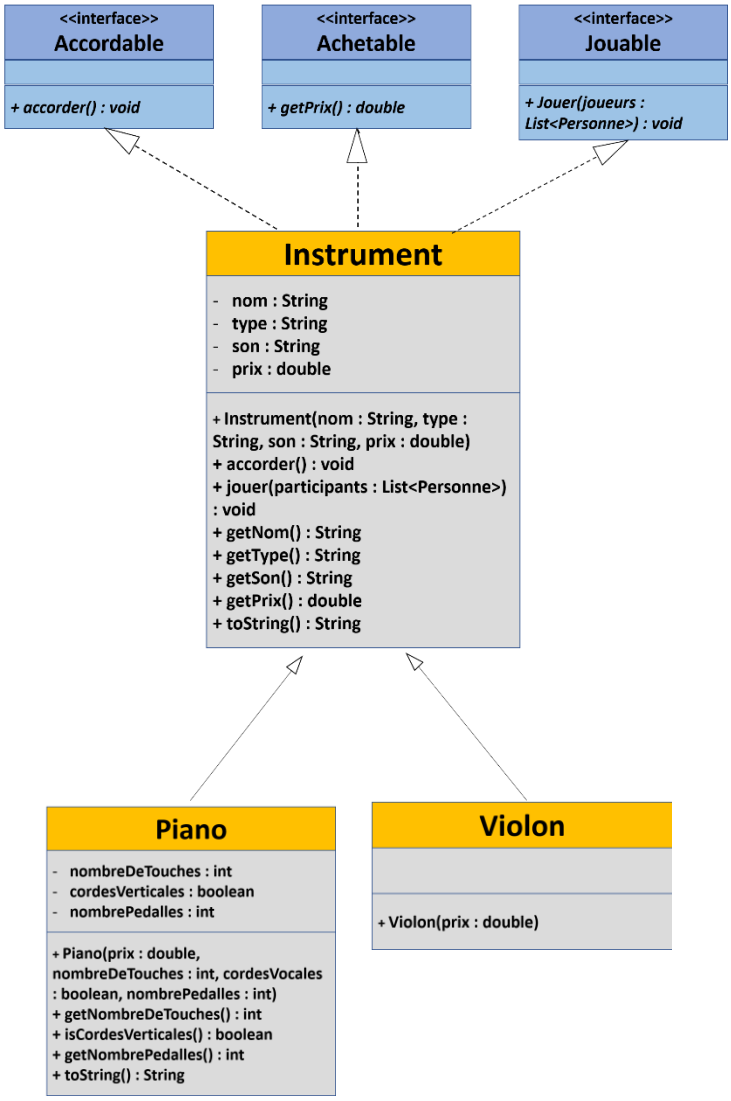
5. **(1 point)** Pour la classe `Personne`, l'attribut `budget` stocke un budget de départ, donnée en entrée du constructeur à chaque utilisation. L'attribut `budgetActuel` stocke le budget actuel d'une personne ; cette valeur est mise à la valeur du budget dans le constructeur.
   Dans la classe Personne on ajoute un attribut `items` de type `List<Achetable>`. Elle sera instanciée à une nouvelle `ArrayList` dans le constructeur.

   Ecrivez une méthode `void acheter(Achetable item)` qui a le fonctionnement suivant : à chaque fois qu'une personne veut acheter un item donné en entrée, s'il s'agit d'un item non-`null`, on vérifie si le budget actuel de la personne suffit pour acheter l'item. Si c'est le cas, on achète l'item : il est rajouté à la liste d'items, puis le budget actuel est diminué par le prix de l'item. Pour chaque erreur (item est `null` ou on n'a pas assez d'argent), un message d'erreur descriptif de l'erreur est affiché.

6. **(1 point)** Pour la question précédente, le paramètre en entrée de la méthode est un objet de type `Achetable`. Qu'est-ce que garantit qu'on peut toujours savoir le prix d'un objet de ce type-là ?

Exercice III (5 points)

Dans cet exercice nous allons écrire du code qui va modéliser des instruments musicaux. Nous allons ensuite créer deux sousclasses comme indiqué dans le diagramme de classe ci-dessous. Vous pouvez supposer que tout le code illustré dans le diagramme de l'annexe A est déjà écrit.



1. **(0.5 points)** La classe `Instrument` fonctionne selon 3 interfaces : `Jouable`, `Accordable` et `Achetable`. Comment pouvez-vous indiquer cela dans votre code ?

2. **(1 point)** Quelles méthodes parmi celles données ci-dessous est-il obligatoire d'implémenter dans la classe Instrument ?
   - ☐ Un constructeur (dont le nom doit être Instrument)
   - ☐ Une méthode `void jouer(List<Personne>)`
   - ☐ Un getter pour chaque attribut
   - ☐ Une méthode avec la signature `double getPrix()`
   - ☐ Une méthode `String toString()`
   - ☐ Un setter pour le son
   - ☐ Une méthode avec la signature `void accorder()`
   - ☐ Une méthode `void jouer()`

3. **(0.5 points)** La classe Instrument a quatre attributs : un nom, un type et un son de type `String`, et son prix, qui est de type `double`. Le nom représente le nom de l'instrument (disons une guitare), le type représente le type d'instrument (par exemple la guitare est un instrument à cordes pincées), tandis que le son représente le son de base qu'il produit (par exemple ti-da).

   Ecrivez une méthode String toString qui retourne :

   `Instrument : <nom>, <type>, <son>, <prix>`

4. **(1 point)** Ecrivez, dans la classe Instrument, une méthode `void accorder()` qui fait juste afficher le message :

   `On accorde l'instrument <nom> : <son>… <son>. Ouf, c'est mieux comme-ça !`

5. **(1 point)** Les classe `Piano` et `Violon` héritent de la classe `Instrument`. Un piano a un certain nombre de touches en fonction de sa taille, il peut être à cordes verticales (piano droit) ou horizontales (piano à queue) et peut avoir un certain nombre de pédales. Ceci est modélisé par les trois attributs propres à la classe `Piano`. Ce constructeur (dont la signature est donnée dans le

diagramme de classe) instancie les attributs hérités d'Instrument aux valeurs suivantes : le nom est `"piano"`, le type est `"cordes frappees"` et le son est `"ding ling"`. Le prix, ainsi que les trois attributs de la classe `Piano` sont instanciés aux valeurs en entrée du constructeur.
Ecrivez ce constructeur.

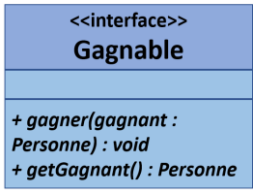
6. **(1 point)** La méthode `String toString()` de la classe `Piano` retourne :
   `Instrument : piano, cordes frappees, ding-ling, <prix>, <nombre de touches>, <cordes verticales ou non>, <nombre de pedales>`

   Ecrivez cette méthode

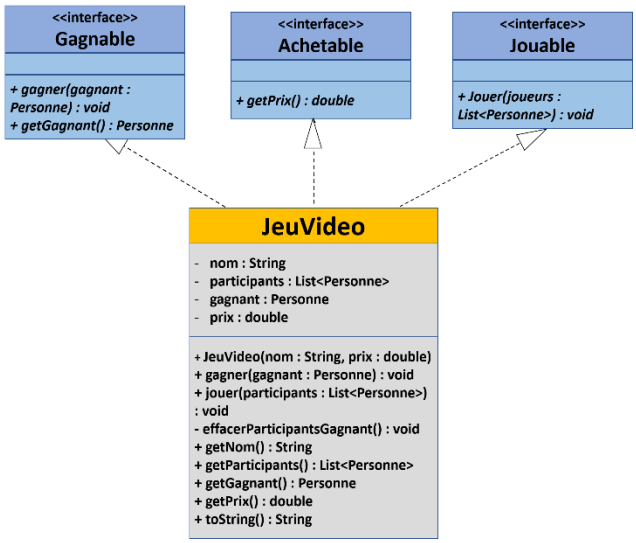
Exercice IV (3.5 points)

Finalement nous aurons des jeux video, qui seront jouables et achetables, mais aussi gagnables -- notamment chaque jeu aura un gagnant. L'interface `Gagnable` est indiquée dans le diagramme de classe ci-dessous, et vous pouvez supposer qu'elle a été écrite.

Vous pouvez également supposer que tout le code des exercices précédents a été bien écrit.



1. **(0.5 points)** La classe `JeuVideo` fonctionne selon les interfaces Jouable, Achetable, Gagnable. Un objet de cette classe a 4 attributs : un nom de type `String`, un prix de type `double`, un objet participants de type `List<Personne>` (qui stocke les participants actuels au jeu et qui est instanciée en tant qu'ArrayList), ainsi qu'un attribut `gagnant` de type `Personne`, qui stocke le dernier gagnant. Le diagramme de classe vous donne une vue d'ensemble sur cette classe.



   La méthode `void effacerParticipantsGagnant()` efface les participants actuels (en utilisant la méthode `clear()` de la classe `ArrayList`) et le gagnant actuel. Ecrivez cette méthode en respectant les spécifications du diagramme de classe.

2. **(1 point)** Vous pouvez désormais supposer que la classe JeuVideo spécifiée dans le diagramme de classe est bien écrite.

On modifie la classe Personne très primitive de l'exercice 2 dans une nouvelle classe Personne dont le diagramme de classe est donné ci-dessus. En dehors des attributs qu'on a discuté précédemment, on y rajoute trois attributs : un attribut jeJoueAu de type Set<Jouable> et un attribut scoresGagnant de type Map<Gagnable, Integer>.

A l'exercice 2, question 5 vous avez écrit une méthode void acheter(Achetable item). Est-ce que cette méthode peut être utilisée tel qu'elle est pour acheter des instruments et des jeux vidéo ou faut-il encore la modifier ? Justifiez votre réponse.

3. **(2 points)** La méthode `void jouer (Jouable jouable, List<Personne> autresJoueurs)` de la classe Personne a le fonctionnement suivant : si le jouable en entrée est non-null, alors il faut utiliser la méthode jouer des objets jouables pour faire jouer l'objet jouable avec les participants suivants : la personne actuelle (`this`) et les autres joueurs dont la liste est donnée en entrée.
   On ajoute le jeu à l'attribut `jeJoueAu`.
   Puis, si le jouable est également un objet de type `Gagnable`, alors : premièrement on cast l'objet jouable à un objet jouableGagnable de type Gagnable. Puis on établit si la `map scoresGagnant` contient la clé `jouableGagnable`. Si ce n'est pas le cas, on ajoute une entrée (`jouableGagnable,0`) à cette map. Dans tous les cas il faut établir si le gagnant du jeu a été la personne actuelle. Si c'est le cas, alors il faut augmenter la valeur correspondante à la clé `jouableGagnable` dans `scoresGagnant` par 1.
   Ecrivez cette méthode.

Exercice V (8 points)

Comme dernier pas il faut juste la mise en scène. Vous pouvez désormais supposer que les classes décrites dans les exercices précédents sont bien écrites (avec la forme complète de la classe Personne. Ce dernier exercice concerne une classe `Examen2019` qui a une méthode `public static void main(String[] args)` contenant le code suivant :

```
public static void main(String[] args) {
	// Personnes :
	final Personne jeanDupont = new Personne("Dupont", "Jean", 1000);
	final Personne michelleBenitez = new Personne("Benitez", "Michelle", 1000);

	// Jeux Video :
	final JeuVideo marioKart = new JeuVideo("Mario Kart", 29.99);
	final Gagnable callOfDuty = new JeuVideo("Call of Duty", 14.99);

	// Instruments :
	Instrument pianoAQueue = new Instrument("piano", "cordes frappees", "ding
ling", 625);
	Jouable pianoDroit = new Piano(455, 120, true, 2);
	Violon violon = new Violon(450);

	// Pizzas :
	Pizza pizzaKebab = new PizzaKebab('M');
	Pizza pizzaPersonnalisee = new PizzaPersonnalisee('L', "creme fraiche", new
String[] {"ricotta", "epinards", "oeufs"});
}
```

1. **(2 points)** Nous allons ajouter les lignes de code suivantes :

```
Accordable[] objetsAccordables = new Accordable[] {pianoAQueue, pianoDroit, violon};
Jouable[] objetsJouables = new Jouable[] {marioKart, pianoAQueue, pianoDroit, violon};
Gagnable[] objetsGagnables = new Gagnable[] {marioKart, callOfDuty};
```

   Pour chaque ligne de code justifiez si elle peut ou non compiler et, si le code ne compile pas, donnez une modification qui pourrait le faire compiler.

2. **(0.75 point)** Les trois tableaux ci-dessus sont un exemple de quel concept ? Ce concept, quelle utilité a-t-il ?

3. **(0.75 point)** Nous voulons afficher les prix des objets `pianoAQueue` et `pianoDroit`. Ecrivez du code qui vous permettrait de faire cela.

4. **(3 points)** Ecrivez du code pour faire les actions suivantes, en utilisant, autant que vous avez la possibilité, les tableaux ci-dessus :
    a. La personne `michelleBenitez` joue au `pianoDroit`, au `pianoAQueue` et au `violon`.
    b. La personne `michelleBenitez` achète l'objet `pianoAQueue`.
    c. La personne `jeanDupont` achète les objets `marioKart`, `callOfDuty` et `pizzaKebab`.
    d. Jean s'entraine : il joue seul au `marioKart` et au `callOfDuty`.
    e. Michelle accorde le piano qu'elle a acheté.
    f. Jean et Michelle jouent ensemble au piano de Michelle.
    g. Jean et Michelle mangent la pizza Kebab
    h. Jean et Michelle jouent au `marioKart`.

5. **(1.5 points)** On affiche la map `scoresGagnant` de `jeanDupont`. Remplissez les contenus que vous pouvez connaître en donnant une limite basse aux valeurs que vous ne connaissez pas. Justifiez les valeurs que vous remplissez.

Annexe A

Diagramme de classe pour le code déjà écrit avant le début de l'exercice 3 :