Langage de programmation orientée objet - Série 3

Objectifs

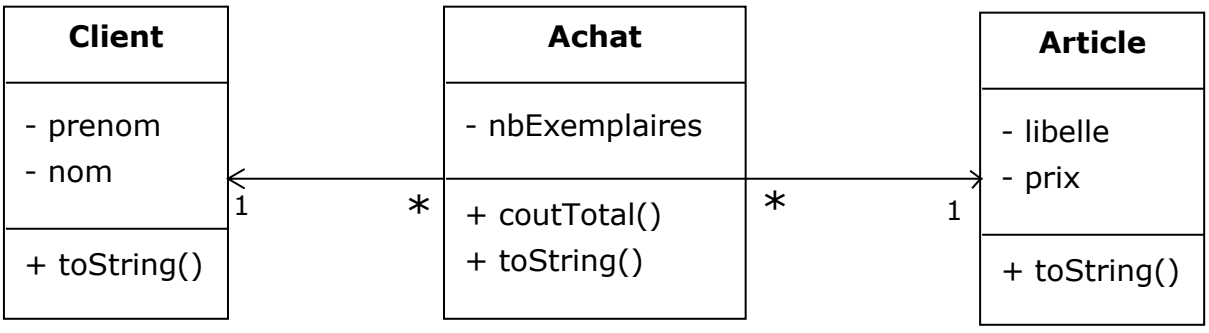
- Manipuler les notions élémentaires : classe et objet, constructeur et méthode, surcharge de constructeur et de méthode, méthode toString, Information Hiding
- Etablir des liens entre objets par l'intermédiaire de variables d'instance
- Par convention, toutes les variables d'instance doivent être privées

Exercice 1 : Achat

Créez un nouveau package appelé `business`.

Vous y placerez les classes permettant de gérer des achats d'articles effectués par des clients. Un achat ne concerne qu'un article mais celui-ci peut être acheté en plusieurs exemplaires.

Diagramme de classes UML:



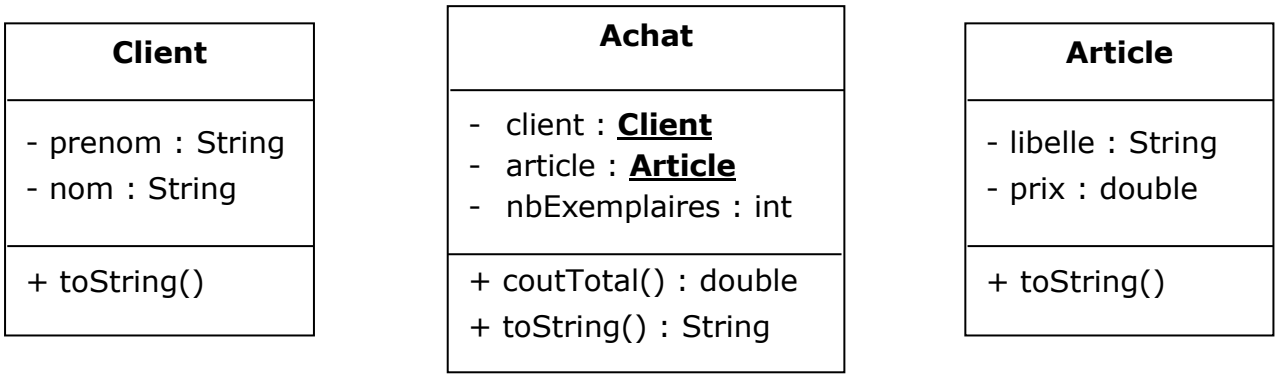
Navigabilité : d'après ce diagramme de classes, à partir d'un achat, on doit pouvoir accéder au client qui effectue l'achat ainsi qu'à l'article acheté. Par contre, à partir d'un client, on ne doit pas accéder à la liste de ses achats. De même, à partir d'un article, on ne doit pas accéder à la liste des achats correspondants.

Liens entre classes

Une association mise en évidence dans un diagramme de classe avec une navigabilité bidirectionnelle est traduite en Java par une variable d'instance dans chacune des classes reliées. S'il s'agit d'une association unidirectionnelle, une variable d'instance est ajoutée dans une seule des deux classes.

Si tout objet d'une classe A ne peut être relié qu'à maximum un seul objet d'une autre classe B (cardinalité maximale à 1), on déclare dans la classe A une variable d'instance du type de la classe reliée B (cf. point 6.1 du syllabus). Si un objet d'une classe A peut être reliée à plusieurs objets d'une autre classe B (cardinalité maximale > 1), on déclare dans la classe A une variable d'instance de type collection (par exemple un tableau) d'objets de type B (cf. exercices ultérieurs).

Si l'on traduit en Java le diagramme de classes ci-dessus, on prévoit donc dans la classe Achat une variable d'instance de type **Client** et une variable d'instance de type **Article**.



Étape 1 : Créer la classe Client

Créez la classe `Client`.

Un client est décrit par son prénom et son nom.

N'oubliez pas de déclarer les variables d'instance avec la protection `private`.

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance.

Prévoyez la méthode `toString` qui retourne le prénom suivi du nom.

Exemple : Alan Turing

Étape 2 : Créer la classe Article

Créez la classe `Article`.

Un article est décrit par son libellé et son prix unitaire.

N'oubliez pas de déclarer les variables d'instance avec la protection `private`.

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance. Le prix d'un article ne peut **pas être négatif** ; certains articles sont cependant gratuits.

Prévoyez la méthode `toString` qui retourne le libellé de l'article suivi entre parenthèses du prix de l'article exprimé en euros.

Exemple : Promo pocket (5.95 euros)

N.B. Si le montant total n'atteint pas 2 euros (ex : 1,99 euro), arrangez-vous pour ne pas afficher de 's' à euros.

Étape 3 : Créez la classe Achat

Créez la classe `Achat`.

Variable d'instance de type classe

Une variable d'instance peut être déclarée avec un type autre que `String` ou un type primitif (`int`, `float`, `double`, `char`, `boolean`…). Le type d'une variable d'instance peut être une classe Java créée par le programmeur.

Un lien peut être établi avec un objet d'une autre classe via une variable d'instance dont le type est cette autre classe.

Déclaration : `NomClasse nomVariableInstance ;`

Soyez donc attentif aux types des variables d'instance `client` et `article`.

Prévoyez un **constructeur** permettant de créer n'importe quel achat. Faites-en sorte que tout achat compte **au moins 1 exemplaire**.

Sachant que dans la plupart des achats, l'article est acheté **en un seul exemplaire**, prévoyez un second constructeur permettant de créer des achats d'exemplaire unique.

Ajoutez les **méthodes** ci-dessous.

- La méthode `coutTotal` calcule le coût total de l'achat.
- La méthode `toString` retourne la chaîne de caractères décrivant un achat en respectant la structure ci-dessous.

Pour un achat en un seul exemplaire :

Appel au toString de Client *Appel au toString d'Article*

⇧ ⇧

Le client … a acheté l'article … en un seul exemplaire
pour un montant total de … euros

⇩

Appel à coutTotal

Pour un achat en plusieurs exemplaires :

Le client … a acheté l'article … en … exemplaires
pour un montant total de … euros

N.B. Si le montant total n'atteint pas 2 euros (ex : 1,99 euro), arrangez-vous pour ne pas afficher de 's' à euros.

Étape 4 : Instancier la classe Achat

Créez la classe `Principale` dans le package `business.`

Dans la méthode `main` de cette classe `Principale`, écrivez les instructions correspondant aux demandes ci-dessous.

- Instanciez un objet appelé `premierClient` de type `Client`.
- Instanciez un objet appelé `premierArticle` de type `Article`.
- Instanciez un objet appelé `premierAchat` de type `Achat`: utilisez les objets `premierClient` et `premierArticle` que vous venez de créer comme arguments lors de l'appel au constructeur de la classe `Achat`; cet achat concerne un article acheté en 3 exemplaires.
- Affichez à la console le coût total de cet objet `premierAchat.`
- Affichez à la console la description de cet objet `premierAchat` ; pour ce faire, utilisez l'instruction `System.out.println(premierAchat)` qui appelle **implicitement** la méthode `toString`.
- A partir de l'objet `premierAchat` que vous avez créé, écrivez les instructions qui répondent aux demandes ci-dessous.

> **Contrainte**
>
> Pour chacun de ces affichages, l'instruction doit impérativement commencer par : `System.out.println(premierAchat.`

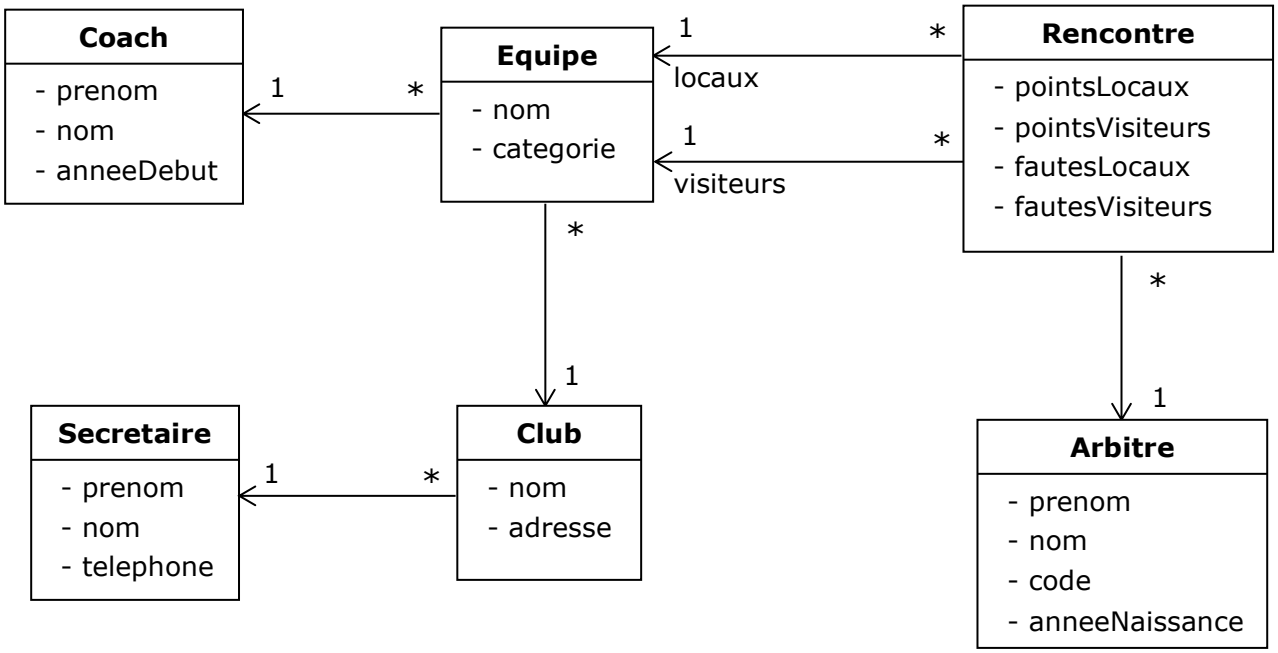
  - Affichez le nombre d'exemplaires achetés de l'objet `premierAchat ;`
  - Affichez le nom du client de l'objet `premierAchat` ;
  - Affichez le libellé de l'article de l'objet `premierAchat.`

- Modifiez le nombre d'exemplaires de l'objet `premierAchat` que vous avez créé ; pour rappel, tout achat doit compter au moins un exemplaire.
- Instanciez un deuxième objet appelé `secondClient` de type `Client`.
- Modifiez le client de l'objet `premierAchat` : remplacez-le par l'objet `secondClient`.
- Affichez à nouveau à la console la description de l'objet `premierAchat`.
- Instanciez un nouvel objet de type `Achat` pour gérer un achat **d'un unique exemplaire** (veillez à utiliser le constructeur le plus adéquat) ; affichez également à la console la description de cet objet.
- Instanciez d'autres objets de type `Achat` et affichez la description de leur état.

Exercice 2 : Tournoi de jeux

Dans le cadre d'un tournoi de jeux, on veut modéliser la gestion de rencontres. Chaque rencontre du tournoi de jeux oppose deux équipes de joueurs ; chaque équipe est affiliée à un club géré par un secrétaire et entraînée par un coach. Une équipe, dite équipe locale, reçoit une équipe de joueurs visiteurs. Au fur et à mesure du déroulement de la rencontre, des points sont attribués à chacune des équipes par un arbitre. Le fair-play étant important, l'arbitre peut également pénaliser une équipe en lui attribuant des fautes.

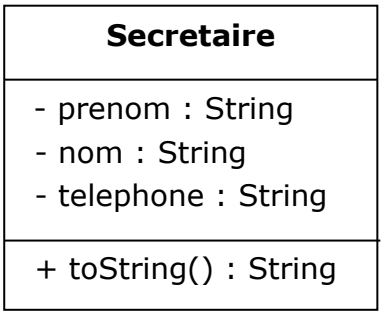
Diagramme de classe :



Créez un nouveau package intitulé `tournois`.

Étape 1 : Classe Secrétaire

Dans le package `tournois`, créez la classe `Secretaire` permettant de gérer des personnes secrétaires d'un club.



Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères composée de la combinaison du prénom et du nom.

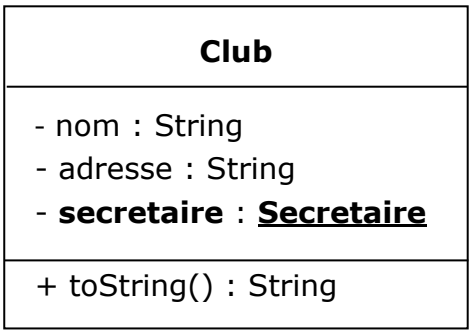
Exemple : Pierre Legrand

Dans le package `tournois`, créez la classe `Principale` qui contient la méthode `main.` Ecrivez-y les instructions pour instanciez deux objets de type `Secretaire` et affichez à la console la description de chacun d'eux (via appel implicite au `toString`).

Étape 2 : Classe Club

Dans le package `tournois`, créez la classe `Club` permettant de gérer des clubs de joueurs.

Un club est décrit par un nom, une adresse et une référence vers un objet de type `Secretaire`.



Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant un club sous le format suivant :

Appel au toString de secretaire

club (nom) (secrétaire : ...)

Exemple : club Fun Game (secrétaire : Pierre Legrand)

Dans la méthode `main` de la classe `Principale`, écrivez les instructions qui répondent aux demandes ci-dessous.

- Instanciez deux objets de type `Club`, chacun ayant un secrétaire différent.
- Affichez à la console la description de chacun des clubs (via appel implicite au `toString`).
- Instanciez un troisième objet de type `Secretaire` et remplacez le secrétaire du premier club que vous avez créé par ce nouvel objet. Affichez de nouveau la description du premier club afin de vérifier que la modification de son secrétaire a bien eu lieu.

Étape 3 : Classe Coach

Dans le package `tournois`, créez la classe `Coach` permettant de gérer des coachs responsables d'équipes de joueurs (entraînement, inscription aux tournois…).

Un coach est décrit par un prénom, un nom et l'année où il a débuté sa carrière de coach.

Coach
- prenom : String - nom : String - anneeDebut : int
+ nbAnneesExperience() : int + toString() : String

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe.

Ajoutez la méthode appelée `nbAnneesExperience` qui retourne le nombre d'années d'expérience du coach.

Date système

Pour récupérer l'année correspondant à la date système (date actuelle), utilisez un objet de type `GregorianCalendar`. Le constructeur **sans argument** permet de créer un objet correspondant à la date du jour. Il suffit ensuite d'accéder en lecture (via la méthode `get`) aux différentes informations de cet objet date. L'argument de cette méthode `get` permet de préciser la partie de la date que l'on souhaite lire (année, mois, jour…). L'argument étant exprimé sous forme de constante (`Calendar.YEAR`), le code est plus aisément compréhensible à la relecture.

```
import java.util.*; // À placer avant la déclaration de la classe
…
// Création d'un objet correspondant à la date système (date du jour)
GregorianCalendar dateDuJour = new GregorianCalendar();
// Récupération de l'année en cours à partir de la date du jour
int anneeEnCours = dateDuJour.get(Calendar.YEAR);
```

Note

Attention, la variable `dateDuJour` doit être déclarée comme variable **locale** à la méthode `nbAnneesExperience`, et non pas comme variable d'instance de la classe `Coach` ! En effet, elle n'est utilisée qu'au sein de la méthode `nbAnneesExperience.`

Adaptez la classe `Coach` de sorte qu'on ne puisse **jamais attribuer à un coach une année de début de carrière supérieure à l'année en cours.** Si l'on tente d'attribuer une année de début de carrière supérieure à l'année en cours, l'année de début de carrière du coach sera initialisée à l'année en cours.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant un coach sous le format suivant :

Appel à la méthode nbAnneesExperience

coach (prenom) (nom) (... années d'expérience)

Exemple : coach Julien Letsgo (8 années d'expérience)

Dans la méthode `main` de la classe `Principale`, écrivez les instructions qui répondent aux demandes ci-dessous.

- Instanciez un premier objet de type `Coach`.
- Instanciez un second objet de type `Coach` en tentant de lui attribuer une année de début de carrière égale à 2100.
- Affichez à la console le nombre d'années d'expérience de chacun d'eux. Vérifiez que le nombre d'années d'expérience du second coach est bien 0.
- Affichez à la console la description de ces deux objets de type Coach (via appel implicite au `toString`).

Étape 4 : Classe Equipe

Dans le package `tournois`, créez la classe `Equipe` permettant de gérer des équipes de joueurs.

Une équipe est décrite par un nom, une catégorie, une référence vers un objet de type `Club` et une référence vers un objet de type `Coach`.

Equipe
- nom : String - categorie : String - **club** : **Club** - **coach** : **Coach**
+ toString() : String

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant une équipe sous le format suivant :

l'équipe (nom) de la catégorie (categorie)
du ... ⇨ *appel au toString du club*
coaché par le ... ⇨ *appel au toString du coach*

Exemple :

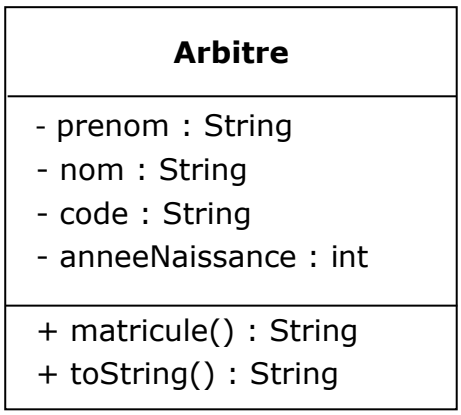
L'équipe les faucons de Valmort de la catégorie jeux de rôles
du club Fun Game (secrétaire : Pierre Legrand)
coaché par le coach Julien Letsgo (8 années d'expérience)

Dans la méthode `main` de la classe `Principale`, écrivez les instructions pour instancier deux objets de type `Equipe`, chacune de ces équipes appartenant à un club différent, et affichez à la console la description de chacun de ces objets (via appel implicite au `toString`).

Étape 5 : Classe Arbitre

Dans le package `tournois`, créez la classe `Arbitre` permettant de gérer les arbitres des rencontres.

Un arbitre est décrit par un prénom, un nom, un code de 3 lettres et une année de naissance.



Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe. **Tout arbitre doit être majeur** ; si on tente d'attribuer à un arbitre une année de naissance invalide (arbitre trop jeune ou pas encore né), on considérera que l'arbitre à 18 ans par défaut.

Ajoutez la méthode appelée `matricule` qui retourne le matricule de l'arbitre formé par la concaténation de l'année de naissance et du code de l'arbitre.

Exemple : Si l'année de naissance de l'arbitre est 1995 et son code est abc, son matricule est 1995abc.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant un arbitre sous le format suivant :

Appel à la méthode matricule

(prenom) (nom) (matricule : ...)

Exemple : Marc Referee (matricule : 1998mlo)

Dans la méthode `main` de la classe `Principale`, écrivez les instructions qui répondent aux demandes ci-dessous.

- Instanciez un objet de type `Arbitre`.

- Affichez à la console le matricule de cet arbitre.
- Affichez à la console sa description (via appel implicite au `toString`).

Étape 6 : Classe Rencontre

Dans le package `tournois`, créez la classe `Rencontre` permettant de gérer les rencontres entre équipes supervisées par un arbitre.

Une rencontre est décrite par :

- une référence vers un objet de type Equipe représentant l'équipe locale ;
- une référence vers un objet de type Equipe représentant l'équipe des visiteurs ;
- le nombre de points de l'équipe locale ;
- le nombre de points de l'équipe des visiteurs ;
- le nombre de fautes de l'équipe locale ;
- le nombre de fautes de l'équipe des visiteurs ;
- une référence vers un objet de type Arbitre.

Prévoyez un **constructeur** permettant d'initialiser toutes les variables d'instance ; les nombres de points et de fautes des équipes ne **peuvent jamais être négatifs.**

Rencontre
- **locaux** : **Equipe** - **visiteurs** : **Equipe** - pointsLocaux : int - pointsVisiteurs : int - fautesLocaux : int - fautesVisiteurs : int - **arbitre : Arbitre**
+ vainqueur() : String + equipeFairPlay() : String + exAequo() : boolean + presentationLocaux() : String + presentationVisiteurs() : String + presentationAdversaires() : String + ajouterPointsAuxLocaux() : void + ajouterPointsAuxVisiteurs() : void + ajouterPointsAuxLocaux(int) : void + ajouterPointsAuxVisiteurs(int) : void + ajouterFauteAuxLocaux() : void + ajouterFauteAuxVisiteurs() : void + ajouterPoints(char) : void + ajouterPoints(char, int) : void + toString() : String

Ajoutez les **méthodes** ci-dessous.

- La méthode `ajouterPointsAuxLocaux` ajoute un point au total des points de l'équipe locale.
- La méthode `ajouterPointsAuxVisiteurs` ajoute un point au total des points de l'équipe des visiteurs.
- La méthode `ajouterFauteAuxLocaux` ajoute une faute à l'équipe locale.
- La méthode `ajouterFauteAuxVisiteurs` ajoute une faute à l'équipe des visiteurs.
- La méthode `vainqueur` retourne le **nom** de l'équipe victorieuse ou la chaîne de caractères "ex æquo" le cas échéant.
- La méthode `equipeFairPlay` retourne le **nom** de l'équipe qui a fait le moins de fautes ou la chaîne de caractères "ex æquo" le cas échéant.
- La méthode `exAequo` retourne `true` si les équipes ont toutes deux le même nombre de points, `false` sinon (écrivez le corps de cette fonction sans utiliser d'alternative !).
- La méthode `presentationLocaux` retourne une chaîne de caractères présentant l'équipe locale, en respectant le format suivant :

  nom de l'équipe locale

  équipe locale ...

  Exemple : équipe locale la guilde de Talrant

- La méthode `presentationVisiteurs` retourne une chaîne de caractères présentant l'équipe des visiteurs, en respectant le format suivant :

  nom de l'équipe des visiteurs

  équipe des visiteurs ...

  Exemple : équipe des visiteurs les faucons de Valmort

*Attention, arrangez-vous pour que les méthodes `vainqueur` et `équipeFairPlay` retournent bien respectivement le **nom** de l'équipe victorieuse ou la plus fair play (ou "ex æquo") et **non pas la description complète de l'équipe correspondante** (pas d'appel aux `toString` sur les équipes). De même, les chaînes de caractères retournées par les méthodes `presentationLocaux` et `presentationVisiteurs` doivent contenir respectivement le **nom** de l'équipe locale et le **nom** de l'équipe des visiteurs et non pas une description complète de l'équipe correspondante.*

Étape 7 : Instancier la classe Rencontre et appeler des méthodes

Placez les instructions qui répondent aux demandes ci-dessous dans la méthode `main` de la classe `Principal`.

- Instanciez un objet appelé `challenge` de type `Rencontre`, à partir des deux objets de type Equipe et de l'objet de type Arbitre que vous avez créés précédemment.
- Ajoutez des points et des fautes aux deux équipes de la rencontre en faisant appel aux méthodes adéquates.
- Affichez à l'écran en faisant appel aux méthodes adéquates :
  - la présentation des deux équipes de la rencontre ;
  - le nom de l'équipe victorieuse ;
  - le nom de l'équipe la plus "fair-play".
- Testez en faisant appel à la méthode adéquate s'il s'agit d'une rencontre où les équipes sont ex aequo.
- Instanciez un second objet de type `Arbitre` et remplacez l'arbitre de l'objet `challenge` par ce nouvel arbitre.

Étape 8 : Autres méthodes

Ajoutez la méthode `presentationAdversaires` qui retourne une chaîne de caractères présentant les deux équipes en respectant le format suivant :

↱ Appel à la méthode presentationLocaux

L'** … **reçoit

l' … .

↳ Appel à la méthode presentationVisiteurs

Exemple : L'équipe locale la guilde de Talrant reçoit l'équipe des visiteurs les faucons de Valmort.

Testez cette méthode sur l'objet `challenge` que vous avez créé.

Ajoutez la méthode appelée `ajouterPoints` qui reçoit en argument un caractère désignant l'équipe ('L' pour l'équipe locale, 'V' pour l'équipe des visiteurs) et qui ajoute un point au total des points de l'équipe correspondante*. N.B. Pour toute autre valeur que 'L' ou 'V' donnée en argument, aucun point n'est ajouté à une équipe.*

Attention : Vous devez impérativement **faire appel aux méthodes existantes** `ajouterPointsAuxLocaux` et `ajouterPointsAuxVisiteurs`.

Testez cette méthode sur l'objet `challenge` que vous avez créé.

Étape 9 : Surcharge du constructeur

Afin de simplifier la création de rencontres qui n'ont pas encore débuté, surchargez le constructeur de manière à ne recevoir que trois arguments (les deux équipes et l'arbitre). Ce constructeur initialisera à 0 les points et le nombre de fautes des deux équipes. Pour rappel, vous devez impérativement appeler le constructeur à 7 arguments **via this(…).**

Testez ensuite ce second constructeur dans la méthode `main` de la classe `Principale` en instanciant de nouvelles rencontres (rencontres qui n'ont pas encore débuté) et en les exploitant ensuite (en appelant des méthodes sur ces nouveaux objets).

Étape 10 : Méthode toString()

Appel implicite à la méthode toString() sur des variables d'instance

Pour rappel, la méthode `toString` est appelée automatiquement chaque fois qu'un nom d'objet est placé là où on attend une chaîne de caractères.

Exemple :

```
NomClasse nomObjet = new NomClasse(…) ;
System.out.println ("blabla " + nomObjet);
```

↳ Appel au `toString` de `NomClasse`

Ce principe peut bien évidemment être appliqué aux variables d'instance : la méthode `toString()` peut être appelée implicitement sur des variables d'instance (cf. point 6.2 su syllabus).

Afin de pouvoir tester les appels implicites en cascade aux différentes méthodes `toString` des classes reliées, créez la méthode `toString` de la classe `Rencontre` afin d'afficher la description de la rencontre en respectant la structure ci-dessous :

Rencontre arbitrée par :
… ⇨ *Appel au toString de l'arbitre*
entre
Locaux :
… ⇨ *Appel au toString de l'équipe locale*
et
Visiteurs :
… ⇨ *Appel au toString de l'équipe des visiteurs*
Vainqueur : … ⇨ *Appel à la méthode vainqueur*

Testez ensuite cette méthode en affichant à la console la description de l'objet `challenge` (via `System.out.println(challenge)`) ainsi que la description des autres objets de type `Rencontre` que vous avez instanciés.

Étape 11 : Surcharge de méthodes

Surchargez les méthodes `ajouterPointsAuxLocaux` et `ajouterPointsAuxVisiteurs`.

- La méthode `ajouterPointsAuxLocaux` **reçoit en argument un nombre de points** et ajoute ce nombre au total des points de l'équipe locale.
- La méthode `ajouterPointsAuxVisiteurs` **reçoit en argument un nombre de points** et ajoute ce nombre au total des points de l'équipe des visiteurs.

Pensez au point de modification unique et modifiez en conséquence les méthodes sans argument.

Testez ensuite le mécanisme de surcharge de méthodes en appelant sur le même objet `challenge` la même méthode d'abord sans argument puis avec argument. Pour vous convaincre de l'effet de ces appels de méthodes, affichez après chaque appel le nombre de points de l'équipe correspondante.

Surchargez également la méthode `ajouterPoints`. Ajoutez une méthode qui reçoit deux arguments : un caractère désignant l'équipe à laquelle il faut ajouter des points et un second argument qui correspond au nombre de points à ajouter à cette équipe. Testez ensuite cette méthode sur l'objet `challenge`.

Pensez également au point de modification unique et modifiez en conséquence les méthodes sans argument.

Étape 12 : Parcourir le graphe des objets reliés à l'objet challenge

A partir de l'objet `challenge` que vous avez créé, écrivez les instructions qui répondent aux demandes ci-dessous.

> **Contrainte**
>
> Chaque instruction doit impérativement débuter par `System.out.println(challenge.`

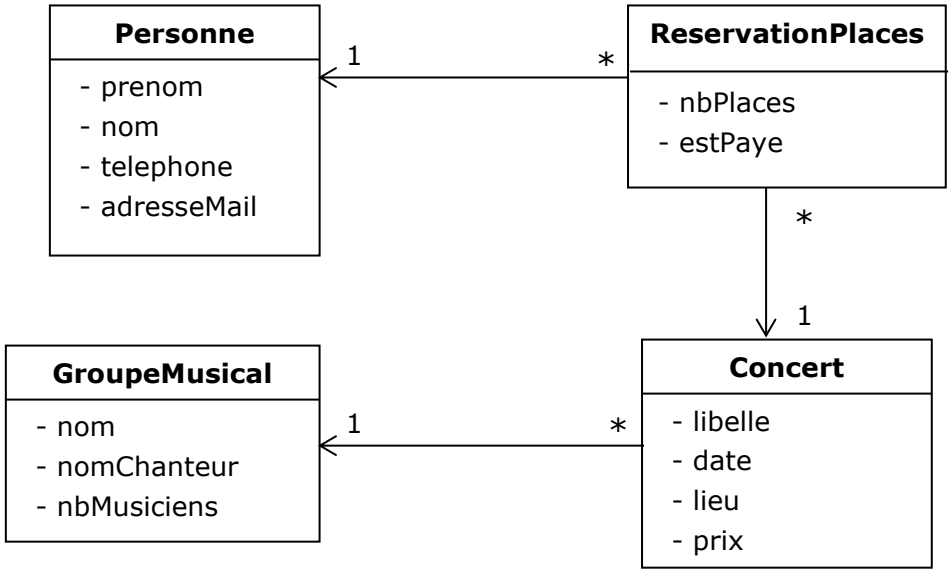
- Affichez la description complète (via appel implicite au `toString`) de **l'arbitre** de l'objet `challenge`.
- Affichez la description complète (`toString`) du **club de l'équipe des visiteurs** de l'objet `challenge`.
- Affichez le **matricule de l'arbitre** de l'objet `challenge`.

- Affichez la description complète (`toString`) du **coach de l'équipe locale** de l'objet `challenge`.
- Affichez la **catégorie de l'équipe locale** de l'objet `challenge`.
- Affichez le **prénom du coach de l'équipe des visiteurs** de l'objet `challenge`.
- Affichez la description complète (`toString`) de **l'équipe des visiteurs** de l'objet `challenge`.
- Affichez l'**adresse du club de l'équipe locale** de l'objet `challenge`.
- Affichez le **numéro de téléphone du secrétaire du club de l'équipe des visiteurs** de l'objet `challenge`.
- Affichez le **nombre d'années d'expérience du coach de l'équipe locale** de l'objet `challenge`.

Exercice 3 : En avant la musique…

Le site web *Music was my first love*, permet aux visiteurs de réserver des places de concert.

Diagramme de classes



Créez un nouveau package intitulé `musique`.

Étape 1 : Classe GroupeMusical

Dans le package `musique`, créez une classe intitulée `GroupeMusical` permettant de gérer des groupes musicaux qui se produisent en concert.

Référez-vous au diagramme de classes pour trouver les variables d'instance à déclarer dans la classe `GroupeMusical`.

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe. Le **nombre de musiciens** ne peut bien évidemment **pas être négatif !**

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant un groupe comparable à l'exemple suivant :

Blue Birds (interprétation : Josh Grant - accompagnement : 5 musiciens)

Dans le package `musique`, créez la classe `Principale` qui contient la méthode `main`. Ecrivez-y les instructions pour instancier un objet de type `GroupeMusical` et affichez à la console sa description (appel au `toString`).

Étape 2 : Classe Concert

Dans le package `musique`, créez une classe intitulée `Concert` permettant de gérer des concerts qui se déroulent en plein air.

Référez-vous au diagramme de classes pour trouver les variables d'instance à déclarer dans la classe `Concert`. Soyez particulièrement attentif à déclarer **ces variables d'instance avec le bon type.**

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe. Le **prix** d'un concert ne peut bien évidemment **pas être négatif** ; on considère cependant que certains concerts peuvent être **gratuits**.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant un concert comparable à l'exemple suivant :

> *le concert intitulé Beach Folk du groupe dénommé*
> *Blue Birds (interprétation : Josh Grant - accompagnement : 5 musiciens)*
> *prévu le 21 juillet 2020 à Knokke*

Pensez au point de modification unique : faites appel à la méthode `toString` de toute classe reliée.

Dans la méthode `main` de la classe `Principale`, écrivez les instructions qui répondent aux demandes ci-dessous.

- Instanciez un objet appelé `ouvertureFestival` de type `Concert`.
- Affichez sa description complète.
- Affichez le libellé, la date et le prix du concert `ouvertureFestival.`
- Modifiez la date, le lieu et le prix du concert `ouvertureFestival.`
- Affichez de nouveau la description de l'objet `ouvertureFestival.`
- A partir de l'objet `ouvertureFestival` que vous avez créé, complétez les instructions ci-dessous pour afficher les informations souhaitées (sans modifier le début de l'instruction proposée).
    - La description complète du groupe musical du concert ouvertureFestival.
      `System.out.println(`**`ouvertureFestival.`**
    - Le nom du chanteur du groupe musical du concert ouvertureFestival.
      `System.out.println(`**`ouvertureFestival.`**
- Instanciez un second objet de type `GroupeMusical` ; remplacez ensuite le groupe musical de l'objet `ouvertureFestival` par ce nouveau groupe.
- Exécutez de nouveau les 2 instructions ci-dessus, à savoir, l'affichage de la description complète du groupe musical de l'objet `ouvertureFestival` ainsi que du nom du chanteur du groupe.

Étape 3 : Classe Personne

Dans le package `musique`, créez une classe intitulée `Personne`.

Référez-vous au diagramme de classes pour trouver les variables d'instance à déclarer dans la classe `Personne`.

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant une personne comparable à l'exemple suivant :

> *Julian Liberry (0499/25.78.42 – jul.liberry@gmail.com)*

Dans la méthode `main` de la classe `Principale`, écrivez les instructions pour instancier un objet de type `Personne` et afficher à la console sa description (appel au `toString`).

Étape 4 : Classe ReservationPlaces

Dans le package `musique`, créez une classe intitulée `ReservationPlaces` permettant de gérer les réservations de places pour des concerts effectuées par des personnes qui visitent le site web *Music was my first love*.

Tout objet de type `ReservationPlaces` correspond à une réservation d'une ou plusieurs place(s) effectuée par un même client pour un concert particulier. Un booléen précise si la réservation est déjà payée ou en attente de paiement.

Référez-vous au diagramme de classes pour trouver les variables d'instance à déclarer dans la classe `ReservationPlaces`. Soyez particulièrement attentif à déclarer **ces variables d'instance avec le bon type.**

Prévoyez un constructeur permettant d'initialiser toutes les variables d'instance de la classe. Toute réservation doit compter **au moins une place.**

Prévoyez la méthode `prixTotal` qui calcule le prix total de la réservation de places en fonction du nombre de places et du prix du concert.

Prévoyez la méthode `toString` qui retourne la chaîne de caractères décrivant une réservation de places comparable à l'exemple suivant :

> *Julian Liberry (0499/25.78.42 – jul.liberry@gmail.com)*
> *a réservé 4 places pour le concert intitulé Beach Folk du groupe dénommé Blue Birds (interprétation : Josh Grant - accompagnement : 5 musiciens) prévu le 21 juillet 2020 à Knokke pour un coût total de 134 euros.*
> *Cette réservation est en attente de paiement.*

Pensez au point de modification unique : faites appel aux méthodes `toString` des classes reliées.

Dans la méthode `main` de la classe `Principale`, écrivez les instructions qui répondent aux demandes ci-dessous.

- Instanciez un objet appelé `reservation` de type `ReservationPlaces`.
- Affichez le prix total de cet objet `reservation`.
- Affichez sa description complète.
- Affichez le nombre de places de l'objet `reservation`.
- Testez la valeur de la variable `estPaye` de l'objet `reservation` : si c'est une réservation payée, affichez "déjà payé", sinon, affichez "en attente de paiement".
- Inversez la valeur de la variable d'instance `estPaye` de l'objet `reservation.`
- Retestez la valeur de la variable `estPaye` de l'objet `reservation` : si c'est une réservation payée, affichez "déjà payé", sinon, affichez " en attente de paiement".
- Modifiez le nombre de places de l'objet `reservation`.
- Affichez de nouveau la description complète de l'objet `reservation`.
- A partir de l'objet `reservation` que vous avez créé, complétez les instructions ci-dessous pour afficher les informations souhaitées.
    - La description complète de l'acheteur de l'objet `reservation.`
      `System.out.println(`**`reservation.`**
    - Le prénom suivi du nom de l'acheteur de l'objet `reservation.`
      `System.out.println(`**`reservation.`**
    - La description complète du concert de l'objet `reservation.`
      `System.out.println(`**`reservation.`**
    - La date du concert de l'objet `reservation.`
      `System.out.println(`**`reservation.`**
    - La description complète du groupe musical du concert de l'objet `reservation.`
      `System.out.println(`**`reservation.`**
    - Le nom du chanteur du groupe musical du concert de l'objet `reservation.`
      `System.out.println(`**`reservation.`**

Étape 5 : Surcharger les constructeurs de ReservationPlaces

Prévoyez les constructeurs supplémentaires ci-dessous.

- Créez un constructeur permettant de réserver une place **unique** à un concert.
- Créez un constructeur permettant de créer une réservation **payée**.
- Créez un constructeur permettant de créer une réservation **payée** d'une place **unique** à un concert.

Testez ensuite ces constructeurs dans la méthode `main` de la classe `Principale` en instanciant de nouvelles réservations (à place unique, payée et à place unique payée) et en affichant leur description.