Exercice 1 — Classes et objets : IMC

Le but de cet exercice est de créer des « patients » qui ont un poids et une taille, et de calculer leur « Indice de Masse Corporelle » (IMC).

Télécharger le programme fourni sur le site du cours et le compléter.
Ecrire le code à fournir entre ces deux commentaires :

```
/*****************************************
* Completez le programme a partir d'ici.
*****************************************/
/*****************************************
* Ne rien modifier apres cette ligne.
*****************************************/
```

Le code fourni

- crée un patient,
- affiche les données du patient ainsi que son IMC.

Ces deux étapes sont répétées deux fois avec des valeurs différentes.

La définition de la classe Patient manque et c'est ce qu'il vous est demandé d'écrire.
Un patient est caractérisé par un poids et une taille. Vous nommerez les attributs correspondants respectivement masse et hauteur.

Par ailleurs, les méthodes spécifiques à un patient sont :

- une méthode init prenant en paramètre deux double, le premier pour initialiser le poids du patient et le second pour initialiser sa taille ;ces données ne seront affectées aux attributs du patient que si elles sont toutes les deux positives ; dans le cas contraire, la taille et le poids du patient seront tous deux initialisés à zéro ; pour simplifier, il n'y a pas d'autre contrôle à faire sur ces données ;
- une méthode afficher permettant d'afficher sur le terminal les caractéristiques du patient en respectant strictement le format suivant :

  Patient : <poids> kg pour <taille> m

  où <poids> est à remplacer par le poids du patient et <taille> par sa taille ; cet affichage sera terminé par un saut de ligne. Vous n'afficherez qu'un chiffre après la virgule pour le poids et la taille.
- une méthode poids retournant le poids du patient ;
- une méthode taille retournant la taille du patient ;
- une méthode imc retournant l'IMC du patient : son poids divisé par le carré de sa taille ; en cas de taille nulle, cette méthode retournera zéro.

Ces méthodes feront partie de l'interface d'utilisation de la classe.

<u>Exemple de déroulement :</u>

```
Patient : 74.5 kg pour 1.8 m
IMC : 24.3265306122449
Patient : 0.0 kg pour 0.0 m
```

Exercice 2 - Constructeurs : Bibliothèque

Le but de cet exercice est de simuler de façon très basique la gestion d'une bibliothèque.
La bibliothèque contient des exemplaires d'œuvres écrites par des auteurs.
Il s'agira de modéliser chacun de ces éléments dans votre programme.

Télécharger le programme fourni sur le site du cours et le compléter.
Ecrire le code à fournir entre ces deux commentaires :

```
/*****************************************
* Completez le programme a partir d'ici.
*****************************************/
/*****************************************
* Ne rien modifier apres cette ligne.
*****************************************/
```

Le code fourni crée des auteurs, des oeuvres de ces auteurs, stocke dans la bibliothèque des exemplaires de ces oeuvres, puis :

- liste tous les exemplaires de la bibliothèque ;
- liste tous les exemplaires écrits en anglais ;
- affiche le nom de tous les auteurs à succès ayant écrit une œuvre dont la bibliothèque stocke un exemplaire ;
- et affiche le nombre d'exemplaires d'une œuvre donnée ;

Un exemple de déroulement possible est fourni plus bas.
Les définitions des classes Auteur, Oeuvre, Exemplaire et Bibliotheque, décrites ci-dessous, manquent et il vous est demandé de les fournir.

La classe **Auteur** : Un auteur est caractérisé par son nom (une String) ainsi qu'une indication permettant de savoir s'il a été primé.
Les méthodes qui sont spécifiques à cette classe et font partie de son interface d'utilisation sont :

- des constructeurs conformes à la méthode main fournie, avec l'ordre suivant pour les paramètres : le nom et l'indication permettant de savoir si l'auteur a été primé ;
- une méthode getNom retournant le nom de l'auteur ;
- une méthode getPrix retournant true si l'auteur a été primé.

La classe **Œuvre** : Une Œuvre est caractérisée par son titre (de type String), (une référence constante à) l'auteur qui l'a rédigée et la langue dans laquelle elle a été rédigée (de type String).
Les méthodes qui sont spécifiques à cette classe et font partie de son interface d'utilisation sont :

- des constructeurs conformes à la méthode main fournie, avec l'ordre suivant pour les paramètres : le titre, l'auteur et la langue. Si la langue n'est pas fournie elle vaudra par défaut "francais" ;
- une méthode getTitre retournant le titre de l'oeuvre ;

- une méthode getAuteur retournant l'auteur (il est toléré ici de retourner directement la référence à l'auteur) ;
- une méthode getLangue retournant la langue de l'oeuvre ;
- et une méthode afficher affichant les caractéristiques de l'oeuvre en respectant strictement le format suivant :
  <titre>, <nom de l'auteur>, en <langue>
  où <titre> est à remplacer par le titre de l'oeuvre, <nom de l'auteur>, par le nom de son auteur et <langue> par sa langue ;

On considère qu'une œuvre n'est pas copiable.

Voir l'exemple de déroulement fourni plus bas pour des exemples d'affichage.

La classe **Exemplaire** : La classe Exemplaire modélise un exemplaire d'une œuvre. Une instance de cette classe est caractérisée par (une référence à) l'œuvre dont elle constitue un exemplaire.
Les méthodes spécifiques à la classe Exemplaire et qui doivent faire partie de son interface d'utilisation sont :

- un constructeur prenant en argument une référence à une oeuvre et affichant un message respectant strictement le format suivant :
  Nouvel exemplaire -> <titre>, <nom de l'auteur>, en <langue>
  suivi d'un saut de ligne ;

- un constructeur de copie affichant un message respectant strictement le format suivant :
  Copie d'un exemplaire de -> <titre>, <nom de l'auteur>, en <langue>
  suivi d'un saut de ligne ;
- une méthode getOeuvre retournant l'oeuvre ;
- et une méthode afficher affichant une description de l'exemplaire respectant strictement le format suivant :
  Un exemplaire de <titre>, <nom de l'auteur>, en <langue>
  sans saut de ligne. La méthode d'affichage de la classe Œuvre sera utilisée pour réaliser cet affichage.

La classe **Bibliotheque** : Une bibliothèque est caractérisée par un nom et contient une liste d'exemplaires.
La liste des exemplaires sera modélisée au moyen d'un tableau dynamique (ArrayList).
Cet attribut devra s'appeler **exemplaires**. Les méthodes spécifiques à la classe Bibliotheque et qui font partie de son interface d'utilisation sont :

- un constructeur conforme à la méthode main fournie et affichant le message :
  La bibliothèque <nom> est ouverte ! suivi d'un saut de ligne,
  où <nom> est à remplacer par le nom de la bibliothèque ;
- une méthode getNom retournant le nom de la bibliothèque ;
- une méthode getNbExemplaires retournant le nombre d'exemplaires contenus dans la liste ;
- une méthode stocker permettant d'ajouter un ou plusieurs exemplaires d'une œuvre dans la bibliothèque ; elle doit être conforme au main fourni,avec l'ordre suivant des

paramètres : la référence à une oeuvre et le nombre n d'exemplaires à ajouter ; cette méthode va ajouter à la liste d'exemplaires de la bibliothèque n exemplaires de l'oeuvre fournie, qu'il s'agit de construire. Si le nombre d'exemplaires n'est pas fourni, cela signifie que sa valeur par défaut est 1. Attention, les exemplaires devront impérativement être ajoutés à la fin du tableau dynamique (méthode add des ArrayList) ;

- une méthode listerExemplaires retournant dans un ArrayList tous les exemplaires d'une œuvre écrite dans une langue donnée ; si aucune langue n'est donnée (chaîne vide), tous les exemplaires de la bibliothèque seront retournés ;Une méthode utilitaire est fournie qui permet ensuite d'afficher le contenu du tableau dynamique retourné par listerExemplaires (voir l'exemple de déroulement fourni plus bas) ;
- une méthode compterExemplaires retournant le nombre d'exemplaires d'une oeuvre donnée passée en paramètre ;
- une méthode afficherAuteur avec un paramètre de type booléen ou sans paramètre, qui affiche les noms des auteurs dont un exemplaire est stocké dans la bibliothèque. Si le booléen est fourni et qu'il vaut true, seuls s'afficheront les noms des auteurs avec un prix ; s'il vaut false seuls les auteurs sans prix s'afficheront. Si le booléen n'est pas fourni, la méthode n'affichera que les noms des auteurs à prix. Les noms appararaîtront autant de fois qu'il y a d'exemplaires d'œuvres écrites par l'auteur. Un saut de ligne sera fait après l'affichage de chaque nom ;

Exemple de déroulement :

```
La bibliotheque municipale est ouverte !
Nouvel exemplaire -> Les Miserables, Victor Hugo, en francais
Nouvel exemplaire -> Les Miserables, Victor Hugo, en francais
Nouvel exemplaire -> L'Homme qui rit, Victor Hugo, en francais
Nouvel exemplaire -> Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Nouvel exemplaire -> Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Nouvel exemplaire -> Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Nouvel exemplaire -> Zazie dans le metro, Raymond Queneau, en francais
Nouvel exemplaire -> The count of Monte-Cristo, Alexandre Dumas, en anglais
La bibliotheque municipale offre
Un exemplaire de Les Miserables, Victor Hugo, en francais
Un exemplaire de Les Miserables, Victor Hugo, en francais
Un exemplaire de L'Homme qui rit, Victor Hugo, en francais
Un exemplaire de Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Un exemplaire de Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Un exemplaire de Le Comte de Monte-Cristo, Alexandre Dumas, en francais
Un exemplaire de Zazie dans le metro, Raymond Queneau, en francais
Un exemplaire de The count of Monte-Cristo, Alexandre Dumas, en anglais
Les exemplaires en anglais sont
Un exemplaire de The count of Monte-Cristo, Alexandre Dumas, en anglais
Les auteurs a succes sont
Raymond Queneau
Il y a 3 exemplaires de Le Comte de Monte-Cristo
```

Exercice 3 – Héritage : Philatélie

Un philatéliste souhaite estimer à quel prix il pourrait vendre ses timbres. Le but de cet exercice est d'écrire un programme lui permettant de le faire.

Télécharger le programme fourni sur le site du cours et le compléter.
Ecrire le code à fournir entre ces deux commentaires :

```
/*****************************************
* Completez le programme a partir d'ici.
*****************************************/
/*****************************************
* Ne rien modifier apres cette ligne.
*****************************************/
```

Le code fourni :

- crée une collection de timbres (rares ou commémoratifs) ;
- et affiche le prix de chaque timbre de cette collection.

La hiérarchie de Timbre manque et c'est ce qu'il vous est demandé d'écrire.
Un timbre est caractérisé par : son code, son année d'émission, son pays d'origine et sa valeur faciale en francs (l'équivalent en francs de la valeur apposée sur le timbre).
Notre philatéliste distingue deux grandes catégories de timbres, se distinguant notamment par les modalités du calcul du prix de vente :

- les timbres rares (classe Rare) : dotés d'un attribut supplémentaire indiquant le nombre d'exemplaires recensés dans le monde ;
- les timbres commémoratifs (Commemoratif) : sans attribut spécifique particulier.

Prix de vente des timbres : Le prix de vente d'un timbre quelconque est sa valeur faciale si le timbre a moins que cinq ans. Sinon le prix de vente vaut la valeur faciale multipliée par l'âge du timbre et par le coefficient 2.5.
Le prix de vente d'un timbre rare part du calcul d'un prix de base : de 600 francs si le nombre d'exemplaires recensés est inférieur à 100, de 400 francs si le nombre d'exemplaires est entre 100 et 1000 et 50 francs sinon. Le prix de vente du timbre est alors donné par la formule prix_base * (age_timbre / 10.0). Les différentes constantes impliquées dans le calcul sont données dans le fichier fourni.
Le prix de vente d'un timbre commémoratif est le double du prix de vente d'un timbre quelconque.

Timbres de base : Les méthodes publiques de la classe Timbre sont :

- des constructeurs conformes à la méthode main() fournie, avec l'ordre suivant pour les paramètres : le code, l'année d'émission, le pays d'origine, et la valeur faciale ; chacun de ces paramètres admet une valeur par défaut :VALEUR_TIMBRE_DEFAUT pour la valeur faciale, PAYS_DEFAUT pour le pays, ANNEE_COURANTE pour l'année et CODE_DEFAUT pour le code ; la construction d'un timbre peut donc se faire avec zéro, un, deux, trois ou quatre paramètres (en respectant l'ordre fourni plus haut) ;
- une méthode vente() retournant le prix de vente sous la forme d'un double ;
- une méthode toString() produisant une représentation d'un Timbre respectant le format suivant :

Timbre de code <code> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs

en une seule ligne où <code> est à remplacer par le code du timbre, <annee> par son année d'émission, <pays>, par son pays d'origine et <valeur faciale>, par sa valeur faciale ;

- la méthode age() retournant l'âge du timbre sous la forme d'un int (différence entre ANNEE_COURANTE et l'année d'émission du timbre) ;
- les getters getCode(), getAnnee(), getPays() et getValeurFaciale().

Timbres rares : Comme méthodes publiques, la classe Rare doit en tout cas fournir :

- des constructeurs conformes à la méthode main() fournie, avec l'ordre suivant pour les paramètres : le code, l'année d'émission, le pays d'origine, la valeur faciale et le nombre d'exemplaires recensés. Le nombre d'exemplaires doit toujours être fourni lors des appels aux constructeurs.
  Les autres paramètres, s'ils ne sont pas fournis, ont les valeurs par défaut indiquées pour les timbres de base ;
- un getter getExemplaires() ;
- une méthode toString() produisant une représentation d'un Rare respectant strictement le format suivant :

  Timbre de code <code> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs

  en une seule ligne puis un saut de ligne suivi de

  Nombre d'exemplaires -> <exemplaires>

  où <code> est à remplacer par le code du timbre, <annee> par son année d'émission, <pays> par son pays d'origine, et <valeur faciale>
  par sa valeur faciale et <exemplaires> par le nombre d'exemplaires recensés.

Il doit être possible de calculer le prix de vente d'un timbre Rare au moyen d'une méthode double vente().

Timbres commémoratifs : Comme méthodes publiques, la classe Commemoratif doit en tout cas fournir :

- des constructeurs conformes à la méthode main() fournie. Ces paramètres, s'ils ne sont pas fournis, ont les valeurs par défaut indiquées pour les timbres de base ;
- une méthode toString() produisant une représentation d'un Commemoratif respectant strictement le format suivant :

  Timbre de code <code> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs

  en une seule ligne puis un saut de ligne suivi de

  Timbre celebrant un evenement

  où <code> est à remplacer par le code du timbre, <annee> par son année d'émission, <pays> par son pays d'origine, et <valeur faciale> par sa valeur faciale.

Il doit être possible de calculer le prix de vente d'un timbre Commemoratif au moyen d'une méthode double vente().

Il vous est donc demandé de coder la hiérarchie de classes découlant de cette description. vous éviterez la duplication de code, le masquage d'attributs et nommerez les classes tels qu'il vous est suggéré de le faire dans l'énoncé.
Par ailleurs vous considérerez que les classes Rare et Commemoratif n'héritent pas l'une de l'autre.

Exemple de déroulement :

```
Timbre de code Guarana-4574 datant de 1960 (provenance Mexique) ayant pour valeur faciale 0.2 francs
Nombre d'exemplaires -> 98
Prix vente : 3360.0 francs
Timbre de code 700eme-501 datant de 2002 (provenance Suisse) ayant pour valeur faciale 1.5 francs
Timbre celebrant un evenement
Prix vente : 105.0 francs
Timbre de code Setchuan-302 datant de 2004 (provenance Chine) ayant pour valeur faciale 0.2 francs
Prix vente : 6.000000000000001 francs
Timbre de code Yoddle-201 datant de 1916 (provenance Suisse) ayant pour valeur faciale 0.8 francs
Nombre d'exemplaires -> 3
        Prix vente : 6000.0 francs
```

Exercice 4 – Polymorphisme : Agence de voyage

Un voyagiste souhaite que vous l'aidiez à gérer ses offres de voyage.

Télécharger le programme fourni sur le site du cours et le compléter.
Ecrire le code à fournir entre ces deux commentaires :

```
/*****************************************
* Completez le programme a partir d'ici.
*****************************************/
/*****************************************
* Ne rien modifier apres cette ligne.
*****************************************/
```

Les options de voyages Notre voyagiste vend des kits de voyage composés de différentes options.
Il s'agit d'abord d'implémenter une classe OptionVoyage permettant de représenter de telles options.

Une option (classe OptionVoyage) est caractérisée par :

- son nom, une chaîne de caractères ;
- et son prix forfaitaire (un double).

La classe OptionVoyage comportera :

- un constructeur initialisant les attributs au moyen de valeurs passées en paramètre et dans un ordre compatible avec le main fourni ;
- une méthode getNom retournant le nom de l'option ;
- une méthode double prix() retournant le prix forfaitaire de l'option ;
- une méthode toString produisant une représentation de l'option sous la forme d'une chaîne de caractères, selon le format suivant :

  <nom> -> <prix> CHF

  où <nom> est le nom de l'option et <prix> est son prix.
  Il vous est demandé d'implémenter la classe OptionVoyage en respectant une bonne encapsulation.

Cette partie de votre programme peut-être testée au moyen de la portion de code comprise entre // TEST 1 et // FIN TEST 1.

Les options de voyage peuvent bien sûr se décliner en différentes sous-classes. Il s'agit ici d'en modéliser deux : les moyens de transport (classe Transport) et le logement pendant le voyage (classe Sejour).

La classe **Sejour :**Une instance de Sejour sera caractérisée par le nombre de nuits (un entier) et le prix par nuit (un double).

Le prix d'un séjour est simplement le nombre de nuits multiplié par le prix par nuit, auquel on ajoutera le prix forfaitaire de l'option.

La classe **Transport :** Une instance de Transport sera caractérisée par un booléen indiquant si le trajet est long.

Le prix du transport vaut la constante TARIF_LONG (1500.0) si le trajet est long et TARIF_BASE (200.0) sinon, auquel on ajoutera le prix forfaitaire de l'option. Les constantes seront publiquement accessibles.

Faites maintenant en sorte que la classe OptionVoyage se spécialise en deux sous-classes : Transport et Sejour répondant à la description précédente.

La hiérarchie de classes sera dotée :

- de constructeurs conformes au main fourni. Les arguments sont dansl'ordre : le nom, le prix forfaitaire et un booléen (valant true si le trajet est long et false sinon) pour les Transport. Les arguments pour le constructeur de Sejour sont dans l'ordre : le nom, le prix forfaitaire, le nombre de nuits et le prix par nuit. Par défaut, un **Transport** a un trajet court.
- de redéfinitions spécifiques de la méthode prix. Ces spécialisations ne contiendront aucune duplication de code et seront utilisables de façon polymorphique.

Cette partie de votre programme peut être testée au moyen de la portion de code comprise entre // TEST 2 et // FIN TEST 2.

Kit de voyage : Le voyagiste vend des kits composés de plusieurs options.
Il vous est demandé de coder une classe KitVoyage comme une «collection hétérogène» de OptionVoyage (un ArrayList).
La classe KitVoyage sera également caractérisée par le départ et la destination du kit (deux String).

La classe KitVoyage sera dotée :

- d'un constructeur compatible avec le main fourni (voir la portion de code entre // TEST 3 et // FIN TEST 3) ;
- d'une méthode double prix() qui calculera le prix du kit comme la somme du prix de toutes ses options ;
- d'une méthode toString, générant une représentation du kit sous la forme d'une String, selon le format suivant :

  Voyage de <depart> à <destination>, avec pour options :
  - <nom option1> -> <prix option1> CHF
  - ....
  - <nom optionN> -> <prix optionN> CHF
  Prix total : <prix du kit> CHF

  où <depart> est le départ du kit, <destination> sa destination et <prix du kit> son prix. La chaîne construite se terminera par \n.
- d'une méthode ajouterOption, compatible avec le main fourni et permettant d'ajouter une OptionVoyage à la collection d'options du kit (les options seront ajoutées en fin de collection). Si l'argument de ajouterOption vaut null, il ne sera pas ajouté à la collection.

- une méthode annuler vidant la collection d'options (utiliser la méthode clear des ArrayList) ;
- une méthode getNbOptions retournant le nombre d'options de voyagedu kit.

Cette partie de votre programme peut être testée au moyen de la portion de code comprise entre // TEST 3 et // FIN TEST 3.

Exemple de déroulement :

```
Test partie 1 :
----------------
Séjour au camping -> 40.0 CHF
Visite guidée : London by night -> 50.0 CHF

Test partie 2 :
----------------
Trajet en car -> 250.0 CHF
Croisière -> 1500.0 CHF
Camping les flots bleus -> 320.0 CHF

Test partie 3 :
----------------
Voyage de Zurich à Paris, avec pour options :
        - Trajet en train -> 250.0 CHF
        - Hotel 3* : Les amandiers -> 540.0 CHF
Prix total : 790.0 CHF

Voyage de Zurich à New York, avec pour options :
        - Trajet en avion -> 1550.0 CHF
        - Hotel 4* : Ambassador Plazza -> 600.0 CHF
Prix total : 2150.0 CHF
```