IG230 – Projet informatique intégré

Informatique de gestion – Bloc 2

Haute-École de Namur-Liège-Luxembourg

Programmation orientée objet avancée
- Série 2 -

Objectifs

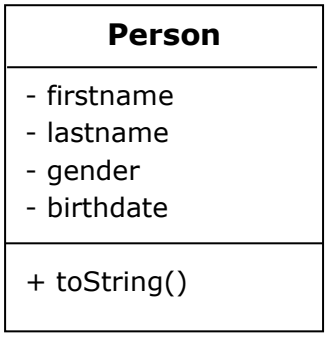
- Gérer les cas d'erreurs via le mécanisme des exceptions
- Utilisez des boîtes de dialogues

Contraintes

- Écrivez le code en **anglais**, que ce soit pour les noms des classes, des variables ou des méthodes.
- Toutes les variables d'instance et les variables de classe doivent être déclarées avec la protection `private`.
- Ne prévoyez **que les getters et setters nécessaires** pour pouvoir exécuter la méthode `main`. *Ne générez donc pas tous les getters et setters automatiquement* !
- Le nombre maximum d'éléments que peut contenir un tableau doit être stocké via une **constante**

Exercice 1 : Première gestion d'exception

Étape 1 : Classe Person



Créez une classe intitulée `Person`**.** Une personne est décrite par un prénom, un nom, un genre (un caractère) et une date de naissance.

Gestion des dates

Utilisez la classe `java.time.LocalDate` pour gérer la date de naissance.

Quelques méthodes utiles :

```
public static LocalDate of(int year,int month,int dayOfMonth)
```
crée une date

```
public static LocalDate now()
```
retourne la date du jour

Prévoyez un constructeur qui permet de créer une personne à partir d'un prénom, un nom, un genre, **une année, un mois et un jour de naissance**. Le constructeur devra donc, à partir d'une année, d'un mois et d'un jour dans le mois créer un objet de type `LocalDate` et l'associer à la variable d'instance `birthdate`.

Les seules valeurs permises pour le genre sont 'm', 'f' et 'x'. Une **exception** (de type `GenderException`) doit être **générée** si on tente d'affecter une autre valeur pour le genre. Cette exception doit être lancée par la méthode setter correspondante (`setGender`).

Aidez-vous du syllabus pour la gestion des exceptions.

Filtrer les arguments reçus par les constructeurs
Les setters peuvent avoir un rôle de filtre dans l'affectation des valeurs aux variables d'instance. Afin de bénéficier de ces filtres, les constructeurs doivent faire appel aux setters. Le constructeur doit donc impérativement faire appel à la méthode `setGender` pour garnir la variable d'instance `gender`. Ce constructeur doit **propager l'exception** éventuellement levée.

Contraintes

À la création de chaque exception de type `GenderException`, générez un message par défaut décrivant l'erreur.

Exemple : La valeur k proposée pour le genre est invalide.

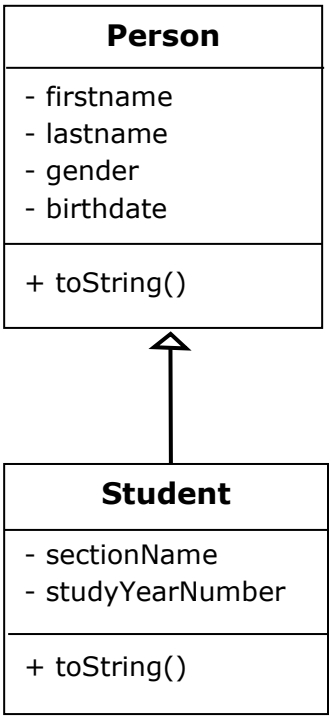
Pour ce faire, prévoyez un constructeur de la classe `GenderException` qui reçoit entre autres un argument de type `String` (message) et qui fait appel au constructeur hérité `Exception(String message).`

Stocker également la valeur (du genre) erronée dans une variable d'instance dans la classe `GenderException` et créez le getter correspondant, au cas où le programmeur qui attrape (`catch`) l'exception souhaite créer son propre message d'erreur (autre que le message par défaut, par exemple des messages d'erreur en plusieurs langues).

Prévoyez également la méthode `toString` dans la classe `Person` qui retourne uniquement le prénom suivi du nom :

ex : *Pierre Legrand*

Étape 2 : Classe Student



Créez une classe intitulée `Student`**.** Un étudiant est une **personne** qui est décrite en outre par le nom de la section dans laquelle il est inscrit ainsi que l'année dans laquelle il est inscrit.

Les seules valeurs permises pour les noms de section sont : *compta, droit, market* et *info*. Une **exception** (de type `SectionException`) doit être **générée** si on tente d'affecter une autre valeur pour la section. Cette exception doit être lancée par la méthode setter correspondante. À la création de chaque exception de type `SectionException`, générez un message par défaut décrivant l'erreur.

Exemple : La valeur danse proposée pour la section est invalide.

Les seules valeurs permises pour l'année dans laquelle un étudiant peut s'inscrire sont 1, 2 ou 3. Une **exception** (de type YearNumberException) doit être **générée** si on tente d'affecter une autre valeur pour l'année. Cette exception doit être lancée par la méthode setter correspondante. À la création de chaque exception de type YearNumberException, générez un message par défaut décrivant l'erreur.

Exemple : La valeur 8 proposée pour l'année est invalide.

Les contraintes pour les classes d'exception sur l'année et la section sont les mêmes que pour la classe `GenderException`.

> **Rappel**
>
> Prévoyez un constructeur qui doit impérativement faire appel aux setters pour garnir les variables d'instance. Ce constructeur doit **propager toutes les exceptions éventuellement générées.**

Prévoyez également la méthode `toString` dans la classe `Student` qui doit **ajouter** à la présentation de la personne (appel au `toString` de `Person`) un message précisant l'année et la section d'inscription.

Veillez à ajouter un "e" à "inscrit" si le genre est "f".

Ex : *Pierre Legrand est inscri**t** en 2e technoInfo.*

*Julie Petit est inscri**te** en 3e droit.*

Étape 3 : Classe Main

Créez une classe appelée `Main` afin :

- **D'essayer** de créer un étudiant et d'afficher à l'écran sa description.
- En cas d'erreur (exception capturée), d'afficher à l'écran le message correspondant à l'erreur.

Exercice 2 : Boîtes de dialogue

Reprenez la classe `Main` :

- En cas de réussite (pas d'erreur) :

Afficher un message de confirmation de l'inscription de l'étudiant **via une boîte de dialogue** (cf la classe `JOptionPane`). Le titre de la boîte de dialogue est : "Confirmation d'inscription" et le contenu du message est la description (appel au `toString`) de l'étudiant.

- En cas d'erreur :

Afficher les messages d'erreurs **via des boîtes de dialogue** dont le titre est "Section non acceptée", "Année d'inscription non acceptée" ou "Valeur pour le genre non acceptée" en fonction de l'erreur et le contenu du message est le message retourné par la méthode `getMessage()`.

Attention

Ajouter l'instruction `System.exit(0)` à la fin de la méthode `main` de la classe `Main` dès que vous utilisez un composant Swing, afin de libérer toutes les ressources et de terminer le processus lié à l'exécution du programme.

Exercice 3 : Obtention des données auprès de l'utilisateur

Étape 1 : Obtention des données d'un nouvel étudiant

Les informations nécessaires pour créer l'étudiant doivent être obtenues auprès de l'utilisateur via de boîtes de dialogue (`JOptionPane`).

Les informations à demander sont : le nom, le prénom, le genre, l'année de naissance, le mois de la date de naissance, le jour de la date de naissance, la section et l'année.

Attention, la méthode `showInputDialog(…)` de la classe `JOptionPane` permet de récupérer un `String`. A vous de transformer les strings en entier (`int`) ou caractère (`char`) si nécessaire.

Suggestion

Chercher les méthodes adéquates pour transformer les `String` en `int`. Faites une recherche dans la classe wrapper `Integer`.

Testez votre programme.

Interprétation des exceptions levées

Dans ce programme, **toutes** les informations pour créer un étudiant sont **demandées** à l'utilisateur **avant d'essayer de créer l'étudiant** correspondant. Or, un seul type d'exception peut être lancée et donc récupérée, car ***dès qu'une exception est levée, le reste de la méthode est abandonn***é. Par conséquent, *même si l'utilisateur a introduit plusieurs valeurs erronées, il ne recevra qu'un seul message le renseignant sur une seule erreur.*

Étape 2 : Création de plusieurs étudiants

Proposez à l'utilisateur de **créer autant d'étudiants qu'il le souhaite**.

Après chaque création d'étudiant, demander à l'utilisateur (via une boîte de dialogue) s'il désire continuer. Bouclez tant que l'utilisateur le désire.

Exercice 4 : Caractéristiques de classe

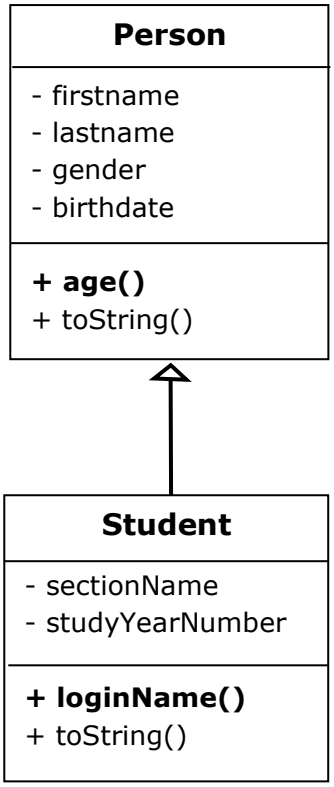
Affichez à la fin du programme (c'est-à-dire dès que l'utilisateur a choisi d'arrêter), le **pourcentage de filles inscrites**.

Contrainte

Interdiction d'utiliser des variables locales dans la méthode `main` de la classe `Main`. Prévoyez une méthode déclarée `static` dans la classe adéquate.

Exercice 5 : Ajout de méthodes

Soit le diagramme de classes complété :



Étape 1 : Login Name

Ajouter la méthode `loginName` dans la classe `Student`.

Cette méthode doit retourner le nom d'utilisateur (login) de l'étudiant qui doit être composé (dans l'ordre) des deux premières lettres de la section, suivi de l'année, suivi du nom complet de l'étudiant et se terminer par l'initiale du prénom.

Par exemple, pour les étudiants *Pierre Legrand inscrit en 2e info* et *Julie Petit inscrite en 3e droit*, les noms d'utilisateur sont respectivement *in2LegrandP* et *dr3PetitJ*.

Suggestion

Consulter la documentation de la classe `String` pour trouver les méthodes permettant de récupérer un caractère ou une partie d'une chaîne de caractères d'une variable de type `String`.

Modifiez ensuite la méthode `toString` de la classe `Student` pour y incorporer le nom d'utilisateur.

Ex : Pierre Legrand est inscrit en 2e info.
Son nom d'utilisateur est in2LegrandP.

Étape 8 : Age

Ajouter la méthode `age` dans la classe `Person.`

Cette méthode doit retourner l'âge de la personne à partir de la date système et de sa date de naissance.

Soyez précis : si la date du jour est le 6 février 2023 et la date de naissance de la personne est le 12 février 2000, cette personne a 22 ans et non 23.

Suggestion

Pour calculer l'intervalle entre deux dates, utilisez la classe `java.time.Period`.

Quelques méthodes utiles :

```
public static Period between(LocalDate startDateInclusive,
                             LocalDate endDateExclusive)
```

Retourne la période (constituée d'un nombre d'années, de mois et de jours) entre les deux dates données

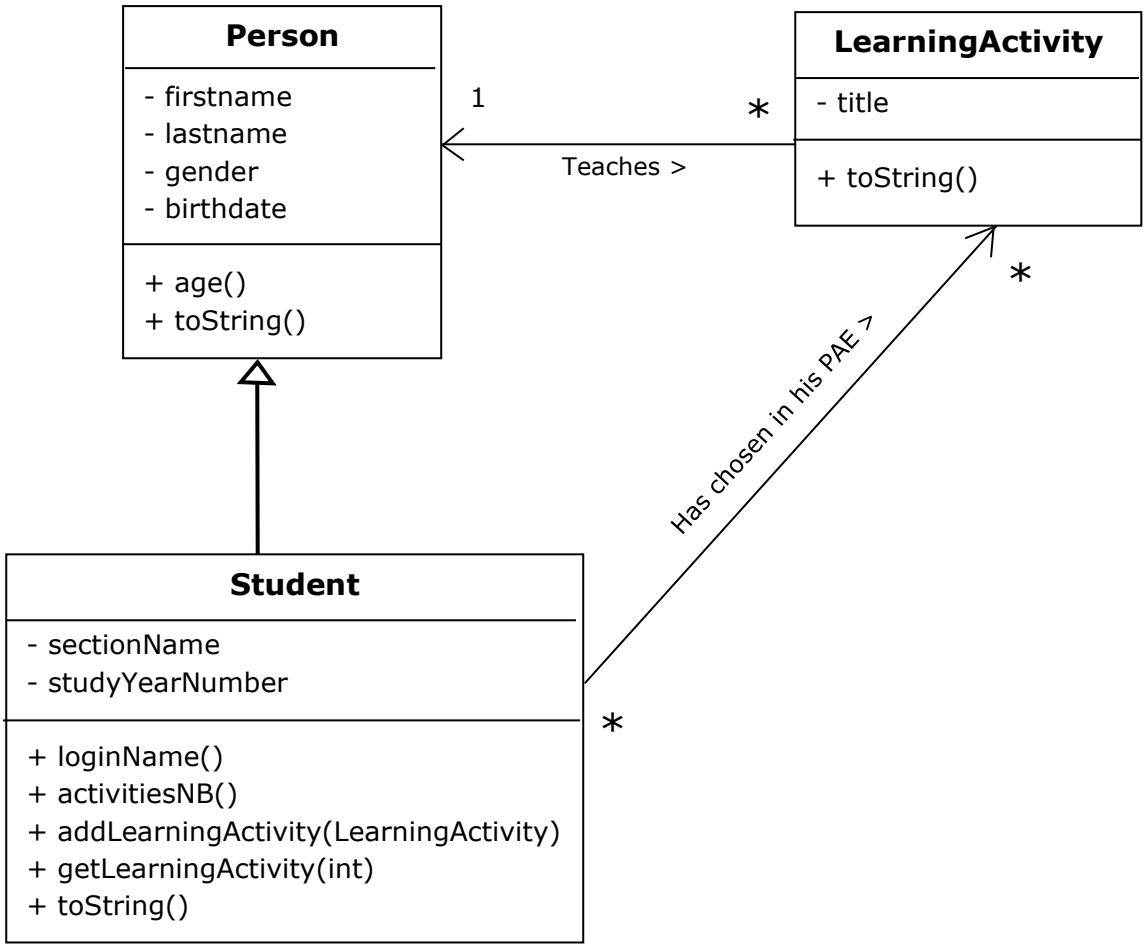
`public int getYears()` retourne le nombre d'années de la période

Modifiez ensuite la méthode `toString` de la classe `Person` pour y incorporer l'âge.

Ex : *Pierre Legrand* ***(18 ans)***

Exercice 6 : Gestion des cas d'erreurs liés aux tableaux

Soit le diagramme de classes suivant :



Étape 1 : Classe LearningActivity

Créez une classe intitulée `LearningActivity`.

Une activité d'apprentissage est décrite par un titre. Complétez, s'il y a lieu, les variables d'instance en fonction du diagramme de classes.

Prévoyez un constructeur ainsi que la méthode `toString` qui retourne la présentation d'une activité d'apprentissage sous le format suivant :

Ex : *l'activité d'apprentissage intitulée Java donnée par Luc Degroot (35 ans)*

Étape 2 : Tableau d'activités d'apprentissage

Adaptez la classe `Student`. Complétez, s'il y a lieu, les variables d'instance en fonction du diagramme de classes.

Suggestion

Pour les besoins de l'exercice, limitons à 5 le nombre maximum d'activités qu'un étudiant peut suivre (cf plus loin). Prévoyez une constante pour stocker le nombre maximum d'activités d'apprentissage qu'un étudiant peut suivre.

Contraintes

Pour les besoins de cet exercice, stockez impérativement les listes d'éléments **dans un tableau** et non dans un objet de type `ArrayList`.

Toutes les variables d'instance (y compris de type tableau) doivent être déclarées `private` !

Adaptez le constructeur : un seul constructeur à prévoir qui permet de créer un étudiant **sans activité d'apprentissage dans son PAE.**

Prévoyez également les méthodes suivantes :

- La méthode `activitiesNb` qui compte le nombre réel d'activités d'apprentissage dans le PAE de l'étudiant.
- La méthode `addLearningActivity` qui ajoute une activité d'apprentissage au PAE de l'étudiant.
- La méthode `getLearningActivity` qui retourne l'activité d'apprentissage du PAE à la position donnée en argument (première position = 1). Ce qui est renvoyé doit être un objet existant (!=null).

Étape 3 : Classe PAEManagement

Créez une classe nouvelle classe appellée `PAEManagement` contenant une méthode `main`.

Cette méthode doit :

- Créer 3 professeurs (personnes) ;
- Créer 3 activités d'apprentissage données par ces professeurs ;
- Créer un étudiant ;

- Ajouter au PAE de l'étudiant les 3 activités d'apprentissage que vous avez précédemment créées ;
- Afficher les activités d'apprentissage du PAE de cet étudiant (bouclez sur toutes les activités d'apprentissage du PAE de l'étudiant) ;
- Afficher la description (`toString`) des professeurs qui donnent les activités d'apprentissage du PAE de cet étudiant (bouclez sur toutes les activités d'apprentissage du PAE de l'étudiant).

Étape 4 : Cas d'erreur lié au débordement du tableau

But

Gérer les cas d'erreurs lorsqu'on tente d'appeler la méthode `addLearningActivity` alors que le tableau des activités d'apprentissage constituant le PAE de l'étudiant est plein.

Créez la classe d'exception `TooManyActivities`.

À la creation d'un objet de type `TooManyActivities`, prévoyez le message par défaut suivant :

Le nombre maximum d'activités d'apprentissage permis est déjà atteint !

Modifiez la méthode `addLearningActivity` afin que soit levée une exception de type `TooManyActivities` si le tableau est plein.

Étape 5 : Cas d'erreur lié à un mauvais numéro d'activité

But

Gérer les cas d'erreurs lorsqu'on tente d'appeler la méthode `getLearningActivity` avec un argument en entrée (position de l'activité d'apprentissage) non valide.

Prévoyez la classe d'exception `BadPositionException` qui comprend deux variables d'instance : la valeur du **mauvais numéro** et le **nombre réel d'activités d'apprentissage** dans le tableau.

À la creation d'un objet de type `BadPositionException`, prévoyez le message par défaut suivant, selon le cas :

- Si la position proposée est <= 0
  (Ex : *Le numéro d'activité d'apprentissage –2 que vous avez proposé est <=0*)
- Si la position proposée ne correspond pas à une activité d'apprentissage suivie par l'étudiant, en rappelant le nombre réel d'activités suivies par l'étudiant
  (Ex : *L'étudiant n'a que 3 activités d'apprentissage dans son PAE. Le numéro 4 que vous avez proposé ne correspond donc pas à une activité d'apprentissage du PAE de cet étudiant*).

Modifiez la méthode `getLearningActivity` afin que soit levée une exception de type `BadPositionException` quand cela est nécessaire.

Étape 6 : Tester les cas d'erreurs

Pour tester ces cas d'erreurs et sachant que l'étudiant que vous avez créé ne suit que 3 activités d'apprentissage, tentez :

- D'afficher l'activité d'apprentissage numéro 4 de cet étudiant ;
- D'afficher l'activité d'apprentissage numéro –1 de cet étudiant ;
- D'afficher l'activité d'apprentissage numéro 0 de cet étudiant ;
- D'ajouter une 4ème, 5ème puis 6ème activité d'apprentissage à l'étudiant.

Exercice 7 : Classes de type collection

Adaptez la classe `Person` : prévoyez une **énumération** pour gérer les valeurs permises pour le genre (`gender`).

L'avantage d'utiliser ici une énumération est qu'il n'y a plus de gestion des cas d'erreur à effectuer, puisque que les seules valeurs permises pour le genre sont les valeurs de l'énumération.

Ce mécanisme peut aussi être appliqué pour les noms de sections et les valeurs des années d'études. Dans la classe `Student`, si les variables d'instance `sectionName` et `studyYearNumber` sont de type énumération, il n'y aura plus de classes d'exception `SectionException` et `YearNumberException` à prévoir.

De même, pour gérer la liste des activités d'apprentissage du PAE, on peut utiliser une classe de type collection comme la classe `ArrayList` plutôt qu'un tableau. Auquel cas, il n'y aura plus d'exception à gérer pour un débordement du tableau ou une tentative d'accès à un élément inexistant. Ces gestions de cas d'erreur sont prises en charge par la classe `ArrayList`.