# Emploi du temps

POnGL, TD3, exceptions.

Nous allons construire des classes pour la gestion d'un emploi du temps hebdomadaire, dont plusieurs méthodes seront susceptibles de lever des exceptions. L'objectif de ce TD est double : organiser ces exceptions, et s'entraîner en prévision de l'examen à écrire du code simple sur papier.

## 1. Lancer une exception

Créer une classe `Horaire` possédant deux attributs `jour` et `heure`, qui sont deux entiers dans des intervalles bien choisis (lundi sera représenté par `0` et dimanche par `6`). Il doit y avoir un constructeur de signature `public Horaire(int j, int h)`, et ce constructeur doit lever une exception `IllegalArgumentException` si les arguments fournis ne correspondent pas à un horaire valide.

Note : la classe `IllegalArgumentException` existe déjà dans la bibliothèque Java. Il s'agit d'une sous-classe de `RuntimeException`, et donc d'une exception hors-contrôle (*unckecked*). Elle possède entre autres un constructeur de signature `public IllegalArgumentException()`.

On demande de plus que les horaires puissent être comparés entre eux via une méthode de signature `public int compareTo(Horaire hor)`, telle que `x.compareTo(y)` renvoie :

- Un nombre négatif si `x` est avant `y`.
- Zéro si `x` et `y` sont égaux.
- Un nombre positif si `x` est après `y`.

## 2. Exceptions et héritage

On définit une nouvelle classe d'exceptions `CréneauInvalide`. On représente un créneau horaire (classe `Créneau`) par une date de début et une date de fin. La classe `Créneau` possède deux constructeurs :

- Le constructeur de signaure `public Créneau(Horaire début, Horaire fin)` prend en argument deux horaires, et lève une exception `CréneauVide` (une sous-classe de `CréneauInvalide`) si l'horaire de `fin` n'est pas strictement après l'horaire de `début`.
- Le constructeur de signature `public Créneau(Horaire début, int duree)` prend en argument un horaire et une durée en heures. Il lève un exception `CréneauVide` si la durée est négative ou nulle, et une exception `Dépassement` (une autre sous-classe de `CréneauInvalide`) si le créneau s'étend au-delà de dimanche 23h59. Une exception de la classe `Dépassement` aura un attribut donnant l'heure de `début` du créneau qu'on a tenté de créer.

On distingue de plus un cas particulier de créneau horaire pour représenter des créneaux de cours (classe `CréneauCours`). De tels créneaux ont comme attribut supplémentaire un entier qui indique leur priorité. On demande à un créneau de cours de ne pas commencer avant 8h, de ne pas finir après 18h, et de ne pas s'étendre sur une nuit ni sur un samedi ou dimanche. Les constructeurs `public CréneauCours(Horaire, Horaire)` et `public CréneauCours(Horaire, int)` doivent lever une exception `NonOuvrable` lorsque l'on essaye de créer un créneau de cours qui n'est pas ouvrable, mais qui est correct par ailleurs. Une exception de la classe `NonOuvrable` aura un attribut contenant ce créneau correct mais non ouvrable (de classe `Créneau`).

Dessiner la hiérarchie des classes `Créneau`, `CréneauCours`, `CréneauInvalide`, `CréneauVide`, `Dépassement` et `NonOuvrable`. Préciser les relations d'héritage, ainsi que les attributs et méthodes de chaque classe, en précisant quelles exceptions peuvent être levées par chaque méthode.

Écrire le code de ces classes, en tirant parti de l'héritage et en utilisant des méthodes auxiliaires pour éviter d'écrire deux fois le même code. Il pourra être judicieux d'avoir une méthode séparée pour calculer l'horaire obtenu quand on ajoute une certaine durée à un horaire de départ. Dans quelle classe définir une telle méthode ?

### 3. Rattraper les exceptions

On veut ajouter à la classe `Créneau` une méthode `public Créneau créneauNonVide(Horaire, Horaire)` qui crée un nouveau créneau à coup sûr, en remettant les horaires de début et de fin dans le bon ordre s'ils ont été fournis en ordre inversé. Faites-le en rattrapant une éventuelle exception `CréneauVide`. Pouvez-vous éviter la déclaration `throws CréneauVide` ? Sinon, comment faire autrement ?

On veut maintenant ajouter à la classe `Créneau` une méthode `public CréneauCours(Horaire, Horaire)` qui rattrape les exceptions `NonOuvrable` pour tenter d'extraire un sous-créneau ouvrable du créneau que l'on a tenté de créer (et qui lève à nouveau une exception `NonOuvrable` si ce n'est pas possible). Par exemple, une tentative de création d'un créneau de cours de 7h à 10h ne produira qu'un créneau de 8h à 10h, tandis qu'une tentative de création d'un créneau de cours de 20h à 23h continuera à lever une exception. Si le créneau demandé s'étale sur plusieurs jours, essayez de faire commencer votre créneau rattrapé au premier jour possible.

Un emploi du temps (classe `EDT`) sera caractérisé par une liste dynamique de créneaux (`ArrayList<Creneau>`). La classe `EDT` doit également contenir une méthode `public void ajouterCreneau(Creneau c)` qui a le comportement suivant :

- Les créneaux sont rangés par ordre chronologique dans la liste.
- Quand on ajoute un créneau qui recouvre des créneaux déjà présents, il faut lever une exception `Conflit` (nouvelle classe d'exceptions à créer), qui contient comme attributs le créneau que l'on a tenté d'ajouter, ainsi que l'indice du premier créneau de la liste qui est en conflit avec lui.

On pourra utiliser la méthode `add(int i, Creneau c)` pour ajouter un créneau `c` à l'indice `i` dans une liste et décaler les éléments suivants.

En cas de conflit, on veut garder les créneaux de plus grande priorité. Ajouter du code pour rattraper une exception de conflit, avec le comportement suivant, en notant `c` le créneau que l'on a tenté d'ajouter et `i` l'indice du premier conflit.

- Regarder parmi les créneaux à partir de l'indice `i` s'il existe un créneau `c'` qui à la fois a une priorité plus élévée que `c` et est en conflit avec `c`.
- Si on a trouvé un tel `c'`, abandonner l'ajout de `c`.
- Sinon, enlever tous les créneaux en conflit avec `c` et ajouter `c` à la place.

On pourra utiliser la méthode `remove(int i)` pour retirer l'élément d'une liste à l'indice `i` et décaler les éléments suivants.