Application : gestion de planning TEAMS

Principe de l'application :

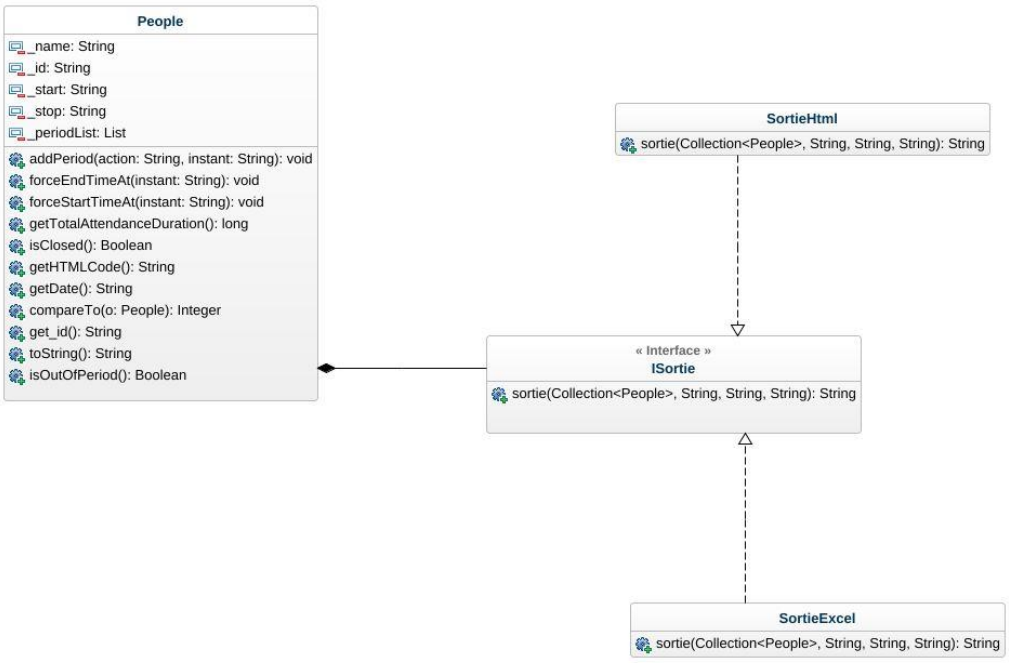
Le but de notre application, et donc de notre projet, est d'utiliser des patterns déjà vus afin de concevoir une petite application de gestion du planning d'une liste de présence récupéré via le logiciel TEAMS. Ce projet consiste à faire du refactoring, c'est à dire à réorganiser le code d'un logiciel. Cette réorganisation est guidée par différents besoins, énoncés ci-dessous. Il s'agit cette année de travailler sur un (petit) outil capable d'extraire d'un fichier texte des informations de façon à les visualiser « proprement » en HTML.

Contrainte : La sortie actuelle est en HTML sur la console.

DP Proposé : **Strategy** est un DP qui permet de définir une famille d'algorithmes, de les mettre dans des classes séparées et de rendre leurs objets interchangeables.

Schéma du DP : https://www.dofactory.com/net/strategy-design-pattern

Remarque : Cela n'est pas pratique en cas de besoin d'extension de l'application comme par exemple, si l'on décide de vouloir un autre type de fichier en sortie. De plus, le mode d'affichage sur la console n'est pas approprié. Le fichier doit être consultable par l'utilisateur. Il y a donc plusieurs possibilités de traitement, qui doivent pouvoir être échangées facilement. On crée donc une interface mère dont les classes filles hériteront la méthode du type de sortie à renvoyer en fonction du choix de l'utilisateur.

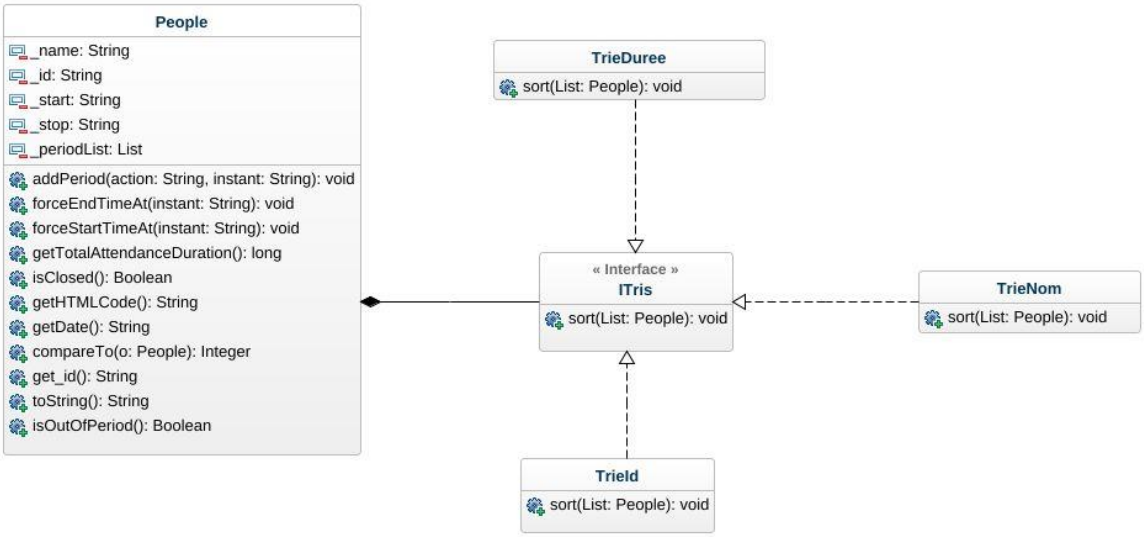


Contrainte : Les données sont, par défaut, triées par durée de connexion croissante.

DP Proposé : **Strategy** est un DP qui permet de définir une famille d'algorithmes, de les mettre dans des classes séparées et de rendre leurs objets interchangeables.

Schéma du DP : https://www.dofactory.com/net/strategy-design-pattern

Remarque : Tout comme lors de la contrainte précédente, nous faisons ici face au même problème d'extension de l'application. Il sera possible d'avoir d'autres options, par ex. par nom d'utilisateur, ou par identifiant. La méthode de tri initiale se trouvait dans la classe People et ne prenait en compte qu'un type de tri (selon la durée de connexion). Afin de pouvoir implémenter d'autres types de tris, il était nécessaire de créer une interface propre à cet algorithme de tri de base afin que les autres classes qui hériteraient de l'interface puissent chacune réaliser des tris avec leur propre implémentation. On crée donc une interface mère dont les classes filles hériteront de la méthode du type de sortie à renvoyer en fonction du choix de l'utilisateur.



Contrainte : Les données peuvent être anonymisées, i.e. les noms, voir les id, ne s'affichent pas.

DP Proposé : **Factory** est un DP de création qui définit une interface pour créer des objets dans une classe mère, mais délègue le choix des types d'objets à créer aux sous-classes.

Schéma du DP : https://www.dofactory.com/net/factory-method-design-pattern

Remarque : Dans cette contrainte, le problème est lié à la création d'objet. Nous cherchons donc à avoir différents types d'affichage lors de la sortie du fichier généré. Cela peut se faire à deux niveaux : les données ne sont pas générées, ou bien elles ne sont pas affichées (neutralisation au niveau du CSS). On peut aussi avoir « sans planning » (juste la liste des connectés). Nous avons fait le choix de procéder à la neutralisation au niveau CSS afin de ne pas afficher les éléments sélectionnés par l'utilisateur dans l'interface graphique. Pour ce faire, nous avons créé une classe abstraite *Hide* dont les classes filles vont hériter des différentes mode d'affichage du fichier généré. Puis, nous avons créé la classe chargée de la création de l'objet Hide en fonction de chaque type d'affichage proposé. Cette dernière sera appelée par le Constructeur relié à notre interface afin de réaliser la tâche.