I. Diagramme de classes d'analyse

D'après le fichier JSON et l'énoncé, on peut dégager les objets métiers suivants

➔**Employee** : représente un employé, caractérisé par un numéro et un type (employé d'administration, employé normal).

À l'employé est associé une carte de temps regroupant ce que dernier a fait durant les 7 jours de la semaine.

Pour un jour de la semaine, l'employé peut travailler sur un ou plusieurs projets.

Nous pouvons mettre en évidence les classes supplémentaires suivantes :

➔**TimeCard** : représente la carte de temps, qui s'étale sur 7 jours (une semaine).

➔**DayOfWeek** : représente un jour de la semaine, qui sera composé d'une liste, éventuellement vide de projets.

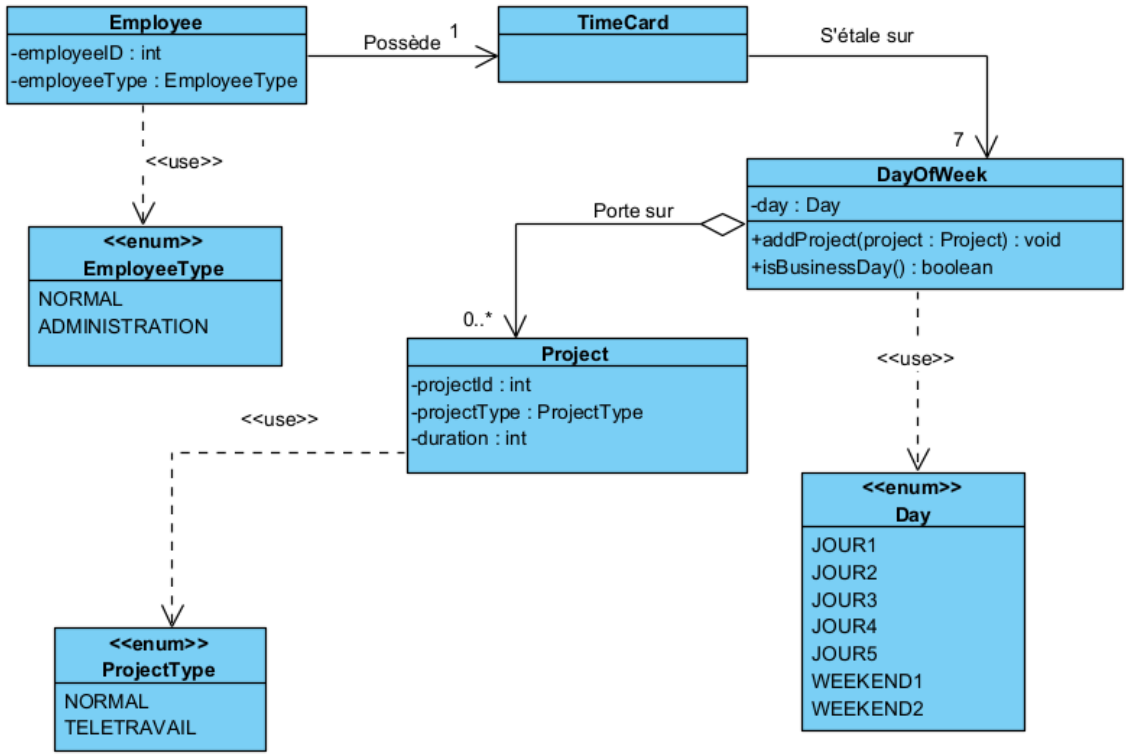
➔**Project** : définit un projet, caractérisé avec un code et un type de projet (télétravail, travail de bureau), ainsi que la durée de travail de l'employé sur ce projet durant le jour de semaine.

Le type d'un employé, le type d'un projet et le jours de la semaine prennent des valeurs fixes. Nous avons ainsi opté pour l'utilisation des énumérations afin de les typer.

➔L'énumération Day est utilisée pour typer l'attribut day (jour de la semaine) de la classe DayOfWeek.

➔L'énumération ProjectType est utilisée pour type l'attribut projectType (type du projet) de la classe Project.

➔L'énumération EmployeeType est utilisée pour typer l'attribut employeeType (type de l'employé) de classe Employee.



➔Toutes les classes métiers appartiennent au package **com.rosemont.model**.

Dans le code Java, nous avons plutôt opté pour des énumérations internes, en tant qu'attribut static des classes pour lesquelles elles sont utilisées pour typer les attributs. Cela nous permet de réduire le nombre de classes, et donc de fichiers source de notre projet.

II. Digramme de classes de conception

Il s'agit ici d'ajouter les classes dites techniques, qui vont réaliser les traitements de l'application. Il s'agit notamment du chargement des données du fichier JSON, de la vérification du respect des règles de l'entreprise ainsi que de l'enregistrement du résultat de cette vérification dans un autre fichier JSON.

Notre diagramme de classes de conception contient les classes suivantes :

➔TimeSheet : classe exécutable de notre programme.

➔Util : classe utilitaire pour le traitement de l'énumération Day afin que l'on puisse utiliser directement les constantes de cette dernière pour l'extraction des données JSON liées aux étiquettes des jours de la semaine.

➔ProcessingJsonData : classe pour le chargement et l'extraction des données JSON depuis le fichier d'entrée, ainsi que l'écriture du résultat dans le fichier JSON de sortie.

➔Classe Message : classe stockant une liste de messages liées aux erreurs de validation de la feuille de temps.

➔Classe Calculation : classe pour les calculs des différents totaux horaires liés au travail de bureau et au télétravail.

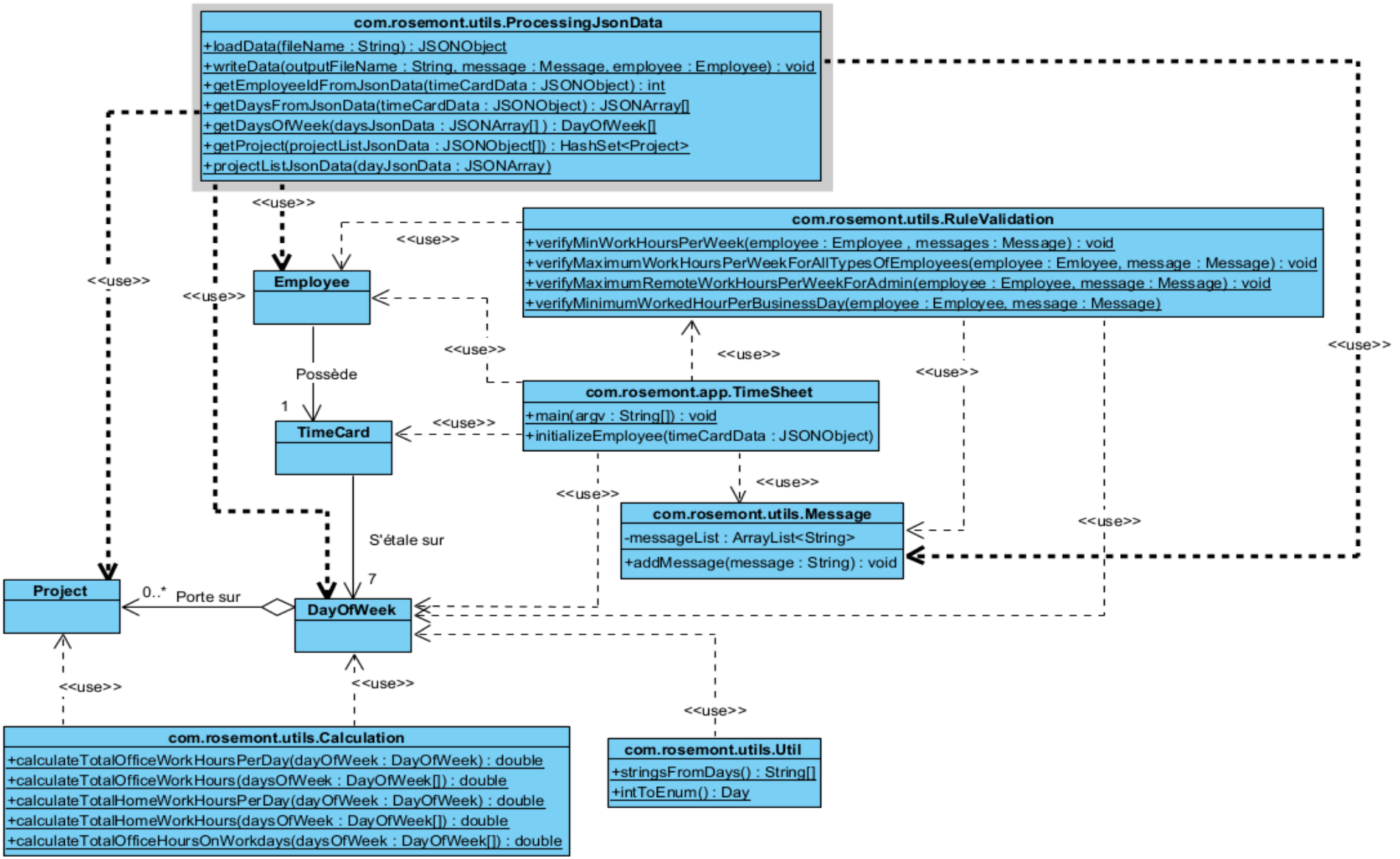
➔Classe RuleValidation : classe vérifiant si la feuille de temps respecte des règles de l'entreprise.

Pour ne pas alourdir le diagramme de classes de conception, nous avons opter pour :

➔L'utilisation d'une représentation simplifiée pour les classes métier (sans attributs et méthodes).

➔L'omission des énumération (celles-ci font partie des classes métier).

➔L'utilisation des noms simples (sans package) pour les classes métier.


com.rosemont.utils.ProcessingJsonData
+loadData(fileName : String) : JSONObject
+writeData(outputFileName : String, message : Message, employee : Employee) : void
+getEmployeeIdFromJsonData(timeCardData : JSONObject) : int
+getDaysFromJsonData(timeCardData : JSONObject) : JSONArray[]
+getDaysOfWeek(daysJsonData : JSONArray[]) : DayOfWeek[]
+getProject(projectListJsonData : JSONObject[]) : HashSet<Project>
+projectListJsonData(dayJsonData : JSONArray)
com.rosemont.utils.RuleValidation
+verifyMinWorkHoursPerWeek(employee : Employee , messages : Message) : void
+verifyMaximumWorkHoursPerWeekForAllTypesOfEmployees(employee : Emloyee, message : Message) : void
+verifyMaximumRemoteWorkHoursPerWeekForAdmin(employee : Employee, message : Message) : void
+verifyMinimumWorkedHourPerBusinessDay(employee : Employee, message : Message)
Employee
Possède
1
TimeCard
com.rosemont.app.TimeSheet
+main(argv : String[]) : void
+initializeEmployee(timeCardData : JSONObject)
com.rosemont.utils.Message
-messageList : ArrayList<String>
+addMessage(message : String) : void
S'étale sur
7
Project
0..*
Porte sur
DayOfWeek
com.rosemont.utils.Calculation
+calculateTotalOfficeWorkHoursPerDay(dayOfWeek : DayOfWeek) : double
+calculateTotalOfficeWorkHours(daysOfWeek : DayOfWeek[]) : double
+calculateTotalHomeWorkHoursPerDay(dayOfWeek : DayOfWeek) : double
+calculateTotalHomeWorkHours(daysOfWeek : DayOfWeek[]) : double
+calculateTotalOfficeHoursOnWorkdays(daysOfWeek : DayOfWeek[]) : double
com.rosemont.utils.Util
+stringsFromDays() : String[]
+intToEnum() : Day
<<use>>

III. Diagramme de package

Les classes de notre programme seront organisées autour de trois packages :

➔**com.rosemont.model :** contient les classes métiers (Employee, DayOfWeek, TimeCard et Project)

➔ **com.rosemont.utils :** contient les classes techniques et utilitaire (ProcessingJsonData, Message, RuleValidation, Util et Calculation).

➔ **com.rosemont.app :** contient la classe exécutable **TimeSheet**.

Le diagramme ci-dessous montre les relations d'import entre ces trois packages :