BELLEGUEULLE Mathieu 25/10/2021

TRAORÉ Moussa

Rapport de premier Jalon

Conception et développement avancé d’application

Pour la partie programmation orientée objet, nous avons créé sept classes, trois classes de gestion de liste et quatre classes pour gérer les données.

# Les classes de gestion de données

On regroupe dans cette partie les classes Contact, Interaction, toDo et Date. Ces classes sont fortement dépendante et donc reliées, une interaction dépend d’un contact et un toDo dépend d’une interaction, c’est pour cela qu’elles disposent en attribut d’un pointeur vers leur instance « propriétaire », c’est-à-dire celle qui les a créés. Les classes dispose Contact et Interaction disposent aussi en attribut une instance de classe de gestion de liste que nous aborderont en seconde partie. Toutes les données sont horodatées, nous devons donc générer et stocker ces dates ce qui impose que les classes aient besoin d’appeler une instance de date pour obtenir une date qui seront ensuite stocker dans des chaînes de caractères pour notamment car elles ne sont pas destinées à changer.

## Classe Contact

La classe soit contenir les informations relatives à un contact (son nom, prénom, entreprise, téléphone ainsi que le lien vers une photo et la date de création) qui sont enregistrer sous forme de chaîne de caractère. De plus, la classe possède un attribut de type GestionInteraction qui sert à contenir les interactions relatives au Contact, dont nous discuterons dans une seconde partie. Pour les fonctions de la classe nous avons des accesseurs en lecture classique et en écriture également mis à part une subtilité, chaque appel de ces fonctions génère une nouvelle interaction témoignant les modifications apportés au contact. On observe également un accesseur en écriture setInterface que l’on retrouve aussi dans Interaction qui sert de transport pour transmettre un pointeur sur une instance d’Interface. La classe possède également deux méthode pour ajouter/enlever des interaction dans la liste qui sont juste une manière de simplifier la création des Interaction qui est ensuite ajouter au sein de l’instance de la classe de gestion de liste. Pour terminer la classe possède une surcharge de l’operateur de comparaison d’égalité qui compare les attributs de deux instances de Contact et de l’opérateur de flux pour l’affichage en console.

## Classe Interface

La classe désigne une interaction avec un contact, elle contient donc un contenu enregistrer sous forme de chaine de caractère et une date générer de manière automatique comme pour les Contacts. La classe est très semblable à la précédente mais l’on peut noter la particularité de la méthode setContenu qu’il convient d’expliquer : à la création de l’interaction, la classe reçoit sous forme de texte plusieurs instructions sous forme d’un texte de plusieurs lignes duquel on distingue deux cas, les instruction dites classique par oppositions aux instructions balisés. Les premières seront mises sans modifications dans la partie contenu de la classe tandis que les secondes seront intégrées sous forme d’instances de classe toDo. La fonction sert à faire la différence entre ces deux type de contenu. Algorithmiquement, tant que l’on n’a pas traiter tout le contenu de la chaine placé en paramètre, on recherche la présence dans la chaîne du mot clef « @todo », si il existe on envoie le texte précédent comme instruction classique dans l’attribut contenu et le texte suivant, jusqu’à la fin de la ligne ou du texte s’il s’agit de l’ultime ligne, sert à la création d’une instance de toDo ajouté ensuite à la liste de l’instance de la classe de gestion de liste.

## Classe toDo

La classe désigne une instruction spécialement balisé, ce type d’instruction contient sémantiquement une instruction à faire avant une date donné. La classe ressemble encore à la précédente au niveau de ses accesseurs et de ses attributs mis à part la classe de gestion de liste inutile. La spécificité de notre classe se trouve dans son constructeur qui décompose la ligne d’instruction. La partie précédent la balise « @date » si présente est enregistrer dans l’attribut contenu de la classe tandis que la partie suivante sert à créer une instance de date particulière. Si cette dernière partie n’est pas présente, la date est initialisée à la date du jour.

## Classe Date

La classe Date sert à symboliser une date pour la transmettre sous forme de chaîne de caractère. Pour ce faire nous utilisons un pointeur sur la structure tm de la librairie ctime. Comme la classe a pour fonction de créer une date, ses fonctions articule dans ce sens. Premièrement, deux constructeurs permettent soit de créer la date du jour, le constructeur sans paramètre, ou de créer une date à partir d’une chaine au format jj/mm/aaaa. Dans ce dernier cas, on utilise la méthode setDate avec comme paramètre une chaine de caractère : les différent champs sont isolés en découpant la chaîne au niveau des « / » on vérifie la cohérence des données et on paramètre une instance de tm pour avoir une structure depuis laquelle on peut obtenir la date sous forme de chaîne de caractère. Cette dernière fonctionnalité est obtenu par la fonction getDateToString() qui récupère les paramètre dans l’instance de tm pointée en attribut pour créer une chaine de caractère représentant la date, cette méthode est importante d’une part car nos dates sont stockées sous forme de chaîne de caractère et d’autre part car bien que l’un de nos constructeur travaille avec une chaîne de caractère en paramètre, ce n’est pas le cas du constructeur par défaut qui lui créer tout de même une date, la date du jour.

Nous avons donc maintenant des classes pour gérer les données et qui sont reliées dans un sens, toDo connait son Interaction et Interaction connait son Contact, il nous reste à les relier dans l’autre sens et comme l’on doit gérer plusieurs instances avec le même propriétaire, nous avons opter pour la création de classes dites de gestion de liste.

# Les classes de gestion de liste

## Classe GestionContact