LANGAGES DE PROGRAMMATION
IFT 3000 NRC 11775
HIVER 2015
Enseignant : Mondher Bouden

Travail pratique 2 (équipe de 2)

À remettre, par Intranet (Portail du cours) avant 17h00 le vendredi 24 avril 2015

1 Énoncé

Nous vous demandons dans ce travail pratique d'implanter **le système d'activités** du TP1 en utilisant le paradigme orienté objet. En effet, il s'agit de reprendre le travail demandé pour le TP1, mais en ajoutant d'autres fonctionnalités. Vous devez d'ailleurs travailler pas seulement avec les activités gratuites (fichier : LOISIR_LIBRE.CSV), mais également avec les activités payantes (fichier : LOISIR_PAYANT.CSV) provenant des données ouvertes de la ville de Québec. D'ailleurs, une activité payante possède un attribut supplémentaire par rapport à une activité gratuite qui est le tarif de base : http://donnees.ville.quebec.qc.ca/donne_details.aspx?jdid=23

2 Travail à faire

Le TP consiste en l'implantation de la structure (*module*) Tp2h15 (voir fichier "TP2-H2015.ml") qui a la signature (*module type*) TP2H15 (voir fichier : "TP2-SIG-H2015.mli"). **Il est à noter qu'il ne faut pas modifier la signature TP2H15.** Dans cette signature, il existe cinq classes (*activite, sysactivites*, *sysactivites_gratuites*, *sysactivites_payantes* et *app_sysactivites*). Les classes *sysactivites_gratuites, sysactivites_payantes* héritent de la classe *sysactivites*. La classe *app_sysactivites* permet de démarrer l'application pouvant chercher pas seulement des activités gratuites comme c'était le cas pour le TP1, mais également des activités payantes se déroulant dans la ville de Québec. Vous pouvez ajouter des fonctions ou des méthodes privées dans le fichier "TP2-H2015.ml". Par contre, votre travail consiste essentiellement à implanter les 14 méthodes suivantes :

1. method afficher_activite : unit **(4 points)**
2. method ajouter_activite : activite → unit **(4 points)**
3. method supprimer_activite : activite → unit **(5 points)**

4. method afficher_systeme_activites : unit **(5 points)**
5. method lire_fichier : in_channel → string → string list list **(5 points)**
6. method trouver_selon_arrondissement : string → activite list **(4 points)**
7. method trouver_selon_type : string → activite list **(4 points)**
8. method lister_arrondissements : string list **(4 points)**
9. method lister_types_activites : string list **(4 points)**
10. method ajouter_liste_activites : string list list → unit **(6 points)**
11. method charger_donnees_sysactivites : string → unit **(7 points)**
12. method trier_activites : int → unit **(12 points)**
13. method sauvegarder_liste_activites : activite list → out_channel → unit **(10 points)**
14. method lancer_systeme_activites : unit **(16 points)**

Vous connaissez normalement ce que fait la majorité de ces méthodes (veuillez consulter l'énoncé ainsi que le corrigé du TP1). Cependant, voici des explications supplémentaires surtout pour les nouvelles méthodes à implanter :

- **trier_activites : int → unit** est une méthode qui permet de trier en ordre croissant les activités qui sont contenues dans le système d'activités. Cette méthode reçoit un entier représentant l'ordre du tri qu'il faut utiliser. Si cet entier est égal à 1, il faut trier les activités selon la date de début ainsi que l'heure de début de l'activité gratuite ou payante. Si cet entier est égal à 2, il faut trier les activités selon la date de fin ainsi que l'heure de fin de l'activité gratuite ou payante. Si cet entier est égal à 3, il faut trier les activités selon le tarif de base de l'activité payante. Ce dernier cas ne s'applique donc pas pour les activités gratuites. Cette méthode doit lancer un message d'erreur si l'entier reçu en paramètre d'entrée est incorrect. Vous devez utiliser dans ce TP *failwith* pour lancer l'exception *failure*. De plus, cette méthode retourne unit car elle fait la mise à jour de la liste d'activités se trouvant dans le système d'activités. Nous vous conseillons d'utiliser la fonction **List.sort** en implantant une fonction **comparer_activites** qui prend deux activités et qui retourne 0 si les deux activités sont égales, 1 si la première activité est plus grande (selon l'ordre mentionné auparavant) que la deuxième et -1 sinon. Voici un exemple d'utilisation de la fonction sort :

  ```
  # List.sort;;
  - : ('a → 'a → int) → 'a list → 'a list = <fun>
  # compare;;
  - : 'a → 'a → int = <fun>
  # List.sort compare [2;5;6;8;4;7;1];;
  - : int list = [1; 2; 4; 5; 6; 7; 8]
  ```

Sinon et afin de comparer les dates et les heures entre-elles, nous vous fournissons une fonction permettant de retourner le nombre de secondes depuis le 1er janvier 1970 jusqu'à une certaine date et une certaine heure précises en utilisant le module Unix (il faut le charger avec #load "unix.cma";;). Voici un exemple d'utilisation de cette fonction qui utilise *decouper_chaine* du TP1:

```
# let ep = retourner_epoque_secondes "2015-04-01" "-" "15:30:00" ":";;
val ep : float = 1427916600.
```

- **sauvegarder_liste_activites : activite list → out_channel → unit** est une méthode qui prend une liste d'activités ainsi qu'un flux déjà ouvert en mode sortie (*out_channel*) et elle écrit ces activités sur ce flux. Le format d'écriture des activités doit être le même qui celui utilisé pour l'affichage de ces activités dans le TP1. Cette méthode ne doit pas fermer le flux. C'est à la méthode appelante qui doit le faire. Il s'agit en fait de la méthode lancer_systeme_activites. De plus, cette méthode doit lancer un message d'erreur si la liste d'activités est vide. Veuillez d'ailleurs consulter les fichiers "je1.ml" (pour tester un système d'activités gratuites) ainsi que "je2.ml" (pour tester un système d'activités payantes) pour produire les mêmes messages d'erreurs. Voici un exemple de code permettant d'écrire dans un fichier texte :

```
# let file = open_out "test.txt";;
val file : out_channel = <abstr>
# let chaine = "Ceci est un test";;
val chaine : string = "Ceci est un test"
# output_string file chaine;;
- : unit = ()
# close_out file;;
- : unit = ()
```

- **lancer_systeme_activites : unit** Vous connaissez normalement l'utilité de cette méthode. Elle permet d'interagir avec l'utilisateur afin de faire une recherche de certains d'activités gratuites ou payantes selon le type (nature) de l'activité ainsi que selon l'arrondissement où se déroule l'activité. Cette méthode peut également maintenant trier ce résultat de recherche selon l'ordre spécifié auparavant dans la méthode trier_activites. Cette méthode peut également sauvegarder le résultat de la recherche dans un fichier texte "Resultat.txt". Veuillez d'ailleurs consulter les fichiers "je1.ml" (pour tester un système d'activités gratuites) ainsi que "je2.ml" (pour tester un système d'activités payantes) pour voir des exemples d'exécution de cette méthode. Cette méthode doit lancer un message d'erreur si l'utilisateur entre un nombre incorrect parmi les

choix possibles. Il est à noter que cette méthode est lancée automatiquement lors de l'instanciation d'un objet de la classe *app_sysactivites*.

3 Démarche à suivre

Puisque le module Tp2h15 contient des méthodes qui ne sont pas encore implantées (c'est à vous de le faire), il ne vous sera pas possible de tester ce module ou de charger le fichier "TP2-H2015.ml", avant de compléter la définition de ces méthodes. La démarche à suivre est la suivante :

- Utiliser un éditeur comme Emacs ou Eclipse, pour compléter l'implantation des méthodes des différentes classes proposées.
- Une fois le module Tp2h15 complété, il vous sera alors possible de le tester et de charger le fichier "TP2-H2015.ml" pour y tester les méthodes qui y sont définies.
- Le test peut se faire en utilisant les fichiers fournis "je1.ml" (pour tester un système d'activités gratuites) et "je2.ml" (pour tester un système d'activités payantes). Dans ces fichiers, il y a les résultats attendus. Vous devez donc vérifier que vous obtiendrez ces mêmes résultats. Il faut savoir que l'exécution du code se trouvant dans "je2.ml" peut prendre quelques secondes étant donné que les activités payantes sont nombreuses et ça peut être lent à charger selon bien sûr la rapidité de votre ordinateur. De plus, il ne faut pas oublier de faire la gestion des erreurs. Vous devez utiliser la fonction **failwith** (string → 'a). D'ailleurs, les fichiers test utilisent **try … with** pour vérifier si l'exception **failure** a été levée.
- Il faut donc mettre ces fichiers ainsi que "TP2-SIG-H2015.mli" dans un même répertoire (éviter les répertoires qui contiennent des espaces ou des accents) et de charger le fichier "je1.ml" ou "je2.ml" avec la commande : **#use "je1.ml";;** ou **#use "je2.ml";; (**il ne faut pas oublier le # avec la commande use).

4 À remettre

À l'aide de l'intranet (Portail du cours), Il faut remettre un fichier zip contenant le fichier "**TP2-H2015.ml**" complété. N'oubliez pas d'indiquer vos noms et matricules dans l'entête de ce fichier. De plus, le fichier zip doit contenir le fichier Excel **NoteTp2.xls** contenant le barème du TP. Il faut ajouter les noms et matricules des membres de votre équipe afin de faciliter le travail du correcteur. Les méthodes implantées doivent être bien indentées. Il est à noter que **10 points** sont donnés pour le respect et la qualité des biens livrables ainsi que pour la structure générale du code.

Bon travail !