TD de *Génie Logiciel Avancé* n° 1
(Correction)

Types abstraits de données

Exercice 1 Dans le centre de loisirs de la ville d'Astride il est possible de faire quatre types d'activité : (1) accéder à la bibliothèque, (2) aller à la piscine, (3) suivre un cours de peinture, (4) jouer au tennis.

L'accès à la bibliothèque est gratuit mais limité aux personnes ayant plus de 12 ans. L'accès à la piscine coûte 15`EUR` par jour. On ne peut pas s'inscrire à la piscine plus de 230 fois dans l'année. Le cours de peinture a lieu une fois par semaine mais son inscription est annuelle et elle coûte 1300`EUR` pour les personnes ayant moins de 12 ans, 1500`EUR` autrement. Enfin, le court de tennis est réservé aux personnes ayant plus de 8 ans et habitant aussi Astride, le prix est de 40`EUR` pour un match double, 60`EUR` pour un match simple. On ne peut pas s'inscrire pour plus de 100 matchs par an. Chaque personne a droit à une seule activité dans l'année. En particulier, pour la piscine et le tennis, chaque personne doit fixer dès le début de l'année le nombre de fois qu'elle souhaite faire cette activité (pour le tennis elle fixera le nombre de matchs simples et le nombre de matchs doubles).

Dans les cinq premières questions suivantes, "définir un type" veut dire donner les constantes, les fonctions de construction, les fonctions d'accès et les fonctions de test à chaque fois que cela est pertinent.

1. Définir un type `age` qui modélise les trois tranches d'âge concernées par les activités du centre.
   Correction :

```
domaine:
age

constantes:
moins8, entre812, plus12: age

operations de test:
est-moins8, est-entre812, est-plus12: age -> bool
```

2. Définir un type `activite_demandee` qui modélise les quatre types de demande d'inscription qu'une personne peut adresser au centre.
   Correction :

```
domaine:
activite-demandee

constantes:
biblio, peinture: activite-demandee

operations de construction:
cons-piscine: {x:N | 1 <= x <= 230} -> activite-demandee
```

```
cons-tennis:  {(x,y):N X N | 1 <= x+y <= 100, 0<=x<=100, 0<=y<=100} -> activite-demandee

operations de construction etendues:
int-to-piscine: N -> activite-demandee
int-int-to-tennis: N X N -> activite-demandee

operations de test:
est-piscine: activite-demandee -> bool
est-biblio: activite-demandee -> bool
est-tennis: activite-demandee -> bool
est-peinture: activite-demandee -> bool

operations d'acces:
nb-fois-piscine: {x: activite-demandee | est-piscine(x)} -> {z:N | 1 <= z <= 230}
nb-simples-tennis: {x: activite-demandee | est-tennis(x)} ->  {z:N | 0 <= z <= 100}
nb-doubles-tennis: {x: activite-demandee | est-tennis(x)} ->  {z:N | 0 <= z <= 100}
```

3. Définir un type `personne` qui modélise le nom, l'âge et la ville d'habitation d'une personne.
   Correction :

```
domaine:
ville

constantes:
astride, autre-ville:ville

operations de test:
est-astride, est-autre-ville: ville -> bool

domaine:
personne

operations de construction:
cons-personne: ville X age X chaines-caracteres -> personne

operations d'acces:
ville-personne:personne  -> vile
age-personne:personne  -> age
nom-personne:personne  -> chainescaracteres
```

4. Définir un type `dossier` qui modélise une demande d'inscription, c'est-à-dire une personne et une activité demandée.
   Correction :

```
domaine:
dossier

operations de construction:
cons-dossier: personne X activite-demandee -> dossier

operations d'acces:
personne-dossier:  dossier -> personne
activite-dossier:  dossier -> activite-demandee
```

5. Définir un type `liste_dossiers` qui modélise une liste de dossiers.
   Correction :

```
domaine:
liste-dossiers

constantes:
liste-vide

operations de construction:
cons-dossier: {(x,y): dossier X liste-dossiers|personne-dossier(x) notin y}-> liste-dossiers

operations de construction etendues:
liste-non-vide-to-liste-dossiers: dossier X liste-dossiers -> liste-dossiers

operations de test:
est-liste-vide: liste-dossiers -> bool
est-liste-non-vide: liste-dossiers ->bool

operations d'acces:
premier-liste: {x: liste-dossiers | est-liste-non-vide(x)} -> dossier
reste-liste : {x: liste-dossiers | est-liste-non-vide(x)} -> liste-dossiers
```

6. Définir une fonction `a_droit_age_ville` telle que `a_droit_age_ville(p,a)` est vrai si et seulement si la personne `p` réunit les conditions permettant de faire l'activité demandée `a`.
   Correction :

```
a-droit-age-ville: personne X activite-demandee -> bool

soit a-droit-age-ville(p,a) =
     selon a
     est-piscine(a): vrai
     est-peinture(a):vrai
     est-biblio(a): est-plus12(age-personne(p))
     est-tennis(a): est-astride(ville-personne(p)) ET not(est-moins8(age-personne(p)))
```

7. Définir une fonction `a_droit_dossier` telle que `a_droit_dossier(d)` est vrai si et seulement si le dossier `d` peut être accepté selon les conditions de l'énoncé.
   Correction :

```
a-droit-dossier: dossier -> bool

soit a-droit-dossier(d) =

     a-droit-age-ville(personne-dossier(d),activite-dossier(d))
```

8. Définir une fonction `prix_annuel_dossier` telle que `prix_annuel_dossier(d)` détermine le prix du dossier `d`.
   Correction :

```
prix-annuel-dossier: dossier -> N

soit prix-annuel-dossier(d) =
     soit p=personne-dossier(d) et a=activite-dossier(d) dans
     selon a
     est-piscine(a): 15 * nb-fois-piscine(a)
     est-peinture(a): si est-plus12(age-personne(p)) alors 1500 sinon 1300
     est-biblio(a): 0
     est-tennis(a): 40*nb-simples-tennis(a) + 60*nb-doubles-tennis(a)
```

9. Définir une fonction `prix_global` telle que `prix_global(l)` détermine le gain du centre de loisirs à partir de la liste de dossiers `l`, en comptabilisant uniquement les dossier valides.
Correction :

```
dossier-unique: personne X liste-dossiers ->bool

dossier-unique(p,l) =
    si est-liste-vide(l)
       alors true
       sinon p = personne-dossier(premier-liste(l)) or
             dossier-unique(p,reste-liste(l))

prix-global: liste-dossiers -> N

soit prix-global(l) =
     si est-liste-vide(l)
        alors 0
        sinon soit d=premier-liste(l) et r=reste-liste(l) dans
              si dossier-unique(personne-dossier(d), r) et a-droit-dossier(d)
                 alors prix-annuel-dossier(d)+ prix-globale(r)
                 sinon prix-globale(r)

(* si la personne demande plusieurs activites, on valide uniquement la derniere *)
```

Exercice 2 Un magasin grossiste de jouets commercialise 3 types de jouets : des billes, des puzzles et des poupées. Le prix d'un puzzle dépend de son nombre de pièces (0,5`EUR` la pièce), le prix d'une bille est de 1`EUR` l'unité, mais il y a une réduction de 2`EUR` par douzaine. Les poupées sont classées en trois catégories, petite taille, moyenne taille, grande taille, et leur prix est fixé en fonction de leur taille : 20`EUR` , 80`EUR` , 170`EUR` respectivement. Chaque client possède un nom et une catégorie. Il y a trois catégories différentes : A, B, C. Les clients de la première catégorie bénéficient d'une réduction de 5% sur les billes et de 3% sur les poupées. Les client de la catégorie B bénéficient d'une réduction de 2% sur les billes et de 7% sur les puzzles, enfin les clients de la troisième catégorie bénéficient d'une réduction de 10% sur les poupées.

1. Définir un type `taille` qui modélise les trois types de tailles de poupées.
   Correction :

```
domaine:
taille

constantes:
petite, moyenne, grande

operations de test:

est-petite:  taille -> bool
est-moyenne: taille -> bool
est-grande:  taille -> bool
```

2. Définir un type `jouet` qui modélise les types de jouets.
   Correction :

```
domaine:
jouet

constantes:
bille

operations de construction:
cons-puzzle: {n:N| 1<=n} -> jouet
cons-poupee: taille -> jouet

operations de construction etendues:
int-to-puzzle: N -> jouet

operations de test:
est-bille: jouet  -> bool
est-puzzle: jouet -> bool
est-poupee: jouet -> bool

operations d'acces:
nb-pieces: {n: jouet | est-puzzle(n))} ->  {n:N| 1<=n}
taile-poupe: {n: jouet | est-poupee(n)} ->  taille
```

3. Définir un type `achat-modele` qui modélise l'achat d'une quantité déterminée d'un type de jouet.
   Correction :

```
domaine:
achat-modele

operations de construction:
cons-achat : jouet * { n:N | n >=1} -> achat-modele

operations de construction etendues:
jouet-int-to-achat: jouet * N -> achat-modele

operations d'acces:
jouet-achat: achat-modele -> jouet
quantite-achat: achat-modele -> { n:N | n >=1}
```

4. Définir un type `categorie` qui modélise les trois différentes catégories d'un client.
   Correction :

```
domaine:
categorie

constantes:
a,b,c

operations de test:

est-a: categorie -> bool
est-b: categorie -> bool
est-c: categorie -> bool
```

5. Définir un type `client` qui modélise un client.
   Correction :

```
domaine:
client

operations de construction:
cons-client: chainecaracteres * categorie -> client

operations d'acces:
nom-client: client  -> chainecaracteres
categorie-client: client  -> categorie
```

6. Définir un type `achat-client` qui modélise un client et tous ses achats.
   Correction :

```
domaine:
liste-achats

constante:
liste-vide

operations de construction:
cons-liste: achat-modele * liste-achats -> liste-achats

operations de test:
est-liste-vide: liste-achats -> bool
```

```
est-liste-non-vide: liste-achats -> bool

operations d'acces:
premier-liste-achats: {n: liste-achats |est-liste- non-vide(n)} -> achat-modele
reste-liste-achats: {n: liste-achats | est-liste-non-vide(n)} -> liste-achats

domaine:
achat-client

operations de construction:
cons-achat-client: client * liste-achats -> achat-client

operations d'acces:
client-achat: achat-client -> client
liste-achat: achat-client -> liste-achats
```

7. Spécifier une fonction `prix-billes` qui renvoie le prix d'une quantité de billes sans considérer les réductions dues aux différentes catégories de clients.
   Correction :

```
prix-billes: N->N

soit prix-billes(e) =
     soit d = e / 12 dans
     soit rest = e - 12*d  dans
          10 * d + rest;;
```

8. Spécifier une fonction `reduction` qui prend un jouet et une catégorie de client en entrée et calcule le coefficient qu'il faut appliquer aux prix de ce jouet en fonction de la catégorie donnée.
   Correction :

```
reduction: categorie X jouet -> R

soit reduction(c,j) =
    si est-bille(j)
       alors si est-a(c)
             alors 0.95
             sinon si est-b(c)
                   alors 0.98
                   sinon 1
       sinon si est-puzzle(j)
             alors si est-a(c)
                   alors 1
                   sinon si  est-b(c)
                         alors 0.93
                         sinon 1
             sinon si est-a(c)
                   alors 0.97
                   sinon si  est-b(c)
                         alors 1
                         sino 0.9
```

9. Spécifier une fonction `prix-modele` qui prend un objet de type `achat-modele` et une catégorie de client et renvoie un prix.
   Correction :

```
prix-modele: achat-modele X categorie -> R

soit prix-modele(a,c) =
   soit j=jouet-achat(a) et e = quantite-achat(a) dans
    si est-billet(j)
    alors reduction(c,j) * prix-billes(e)
    sinon si est-puzzle(j)
          alors reduction(c,j) * 0.5 * n * e
          sinon soit t = taille-poupee(j) dans
                soit p = (si est-petite(t)
                             alors 20
                             sinon si est-moyenne(t)
                                   alors 80
                                   sinon 170) dans
                reduction(c,j) * p * e
```

10. Spécifier une fonction `prix-total` qui renvoie le prix de la liste de tous les achats d'un client.
    Correction :

```
prix-total: liste-achats -> R

soit prix-total(c,l) =
    si est-liste-vide(l)
       alors 0
       sinon prix-modele(p, categorie-client(c)) + prix-total(c,r)
```