# TP - Structures de données, interface et implémentation

## Exercice 1 La structure de données `Message`

1. Implémenter toutes les fonctions de l'interface de l'exercice 7 du TD en utilisant la structure `list` pour représenter un message.
   - Vous testerez votre implémentation en codant le programme python de l'exercice 7.
   - Pour implémenter la fonction `sendMsg(msg)`, vous vous contenterez d'afficher un message indiquant le message à bien été envoyé vers le destinataire.
2. Implémenter toutes les fonctions de l'interface de l'exercice 7 du TD en utilisant la structure `dictionnaire` pour représenter un message.
   - Vous testerez votre implémentation en codant le programme python de l'exercice 7. **Ce programme doit être le même que celui de la question 1**. En effet, l'interface reste la même et son utilisation est indépendante de l'implémentation choisie.
   - Pour implémenter la fonction `sendMsg(msg)`, vous vous contenterez d'afficher un message indiquant le message à bien été envoyé vers le destinataire.

## Exercice 2 La structure de données `Temps`

On dispose d'une structure de données appelée `Temps` permettant de stocker des temps (données en heures, minutes, secondes). Voici les opérations que l'on souhaite effectuer sur le type abstrait `Temps` :

- Créer un temps
- Accéder aux heures, minutes, secondes d'un temps
- Ajouter, soustraire deux temps
- Vérifier si deux temps sont égaux ou non
- Afficher un temps sous forme d'une chaîne de caractères

## Partie A - Interface

On peut spécifier les opérations de cette structure de données en proposant l'interface suivante :

- `Temps(h, m, s)` : crée un élément de type `Temps` à partir de trois entiers `h` (heures), `m` (minutes) et `s` (secondes).
- `heures(t)` : accès aux heures du temps `t` (renvoie un entier).
- `minutes(t)` : accès aux minutes du temps `t` (renvoie un entier).
- `secondes(t)` : accès aux secondes du temps `t` (renvoie un entier).
- `ajouter(t1, t2)` : renvoie un nouveau temps correspondant à la somme des temps `t1` et `t2`.
- `soustraire(t1, t2)` : renvoie un nouveau temps correspondant à la différence de `t1` et `t2`.
- `egal(t1, t2)` : renvoie `Vrai` si les deux temps `t1` et `t2` sont égaux, `Faux` sinon.
- `afficher(t)` : affiche le temps `t` sous forme d'une chaîne de caractères h :m :s.

Par abus de notation, on a aussi noté `Temps` l'opération de construction du type abstrait `Temps`.

1. On crée deux temps.

```
t1 = Temps(2, 18, 3)
t2 = Temps(1, 50, 12)
```

   Quelles sont les valeurs des instructions `heures(t1)`, `minutes(t2)` et `secondes(t1) + secondes(t2)` ?

2. (a) Quelle instruction faut-il écrire pour créer un nouveau temps `t` qui est la somme des temps `t1` et `t2` ?
   (b) Que vaut alors le temps `t` ?
   (c) Qu'affiche alors l'instruction `afficher(t)` ?
3. Quelle est la valeur renvoyée par l'instruction suivante ? `egal(t, Temps(4, 8, 15))`
4. Lors d'une course de relais par équipe, les trois membres d'une équipe ont mis respectivement les temps suivants pour boucler leurs relais respectifs :
   - 0 h 51 min 14 sec
   - 1 h 02 min 53 sec
   - 0 h 56 min 31 sec

   Quelles instructions permettent d'afficher le temps total mis par l'équipe pour terminer la course ?
5. Le dernier relayeur est arrivé précisément à 18 heures 15 minutes et 3 secondes. Quelles instructions permettent d'afficher l'heure de départ du premier relayeur ?

## Partie 2 - Implémentation

On choisit ici d'utiliser un tableau pour représenter le type de données abstrait `Temps`.

1. A partir de l'implementation suivante, expliquer la solution choisie pour effectuer les calculs avec des données de type `Temps`.
2. Que fait la fonction `int_vers_temps(secondes)` ?
3. Que fait la fonction `temps_vers_int(t)` ?
4. Compléter l'implémentation et tester le code avec les exemples de la partie 1.

```
def Temps(h, m, s):
    """Entier x Entier x Entier --> Temps"""
    return [h, m, s]

def heures(t):
    """Temps --> Entier"""
    # A COMPLETER

def minutes(t):
    """Temps --> Entier"""
    # A COMPLETER

def secondes(t):
    """Temps --> Entier"""
    # A COMPLETER

def ajouter(t1, t2):
    """Temps x Temps --> Temps"""
    # A COMPLETER

def soustraire(t1, t2):
    """temps x temps --> temps
    Renvoie la difference t1 - t2, ou t1 >= t2"""
    assert t1 >= t2, "le premier temps doit etre superieur au second"
    secondes = temps_vers_int(t1) - temps_vers_int(t2)
    return int_vers_temps(secondes)

def egal(t1, t2):
    # A COMPLETER

def afficher(t):
    print(str(t[0]) + ":" + str(t[1]) + ":" + str(t[2]))

def temps_vers_int(t):
    # A COMPLETER

def int_vers_temps(secondes):
    m, s = divmod(secondes, 60)
    h, m = divmod(m, 60)
    return [h, m, s]
```

## **Exercice** 3 **La structure de données `Point`**

On dispose d'une structure de données appelée `Point` permettant de représenter des points à afficher dans un programme de dessin. Chaque point a une abscisse et une ordonnée et le repère est orthonormé.

Voici les opérations que l'on souhaite effectuer sur le type abstrait `Point` :

- Créer un point
- Accéder à son abscisse et à son ordonnée
- Modifier les coordonnées d'un point
- Déterminer le milieu d'un segment
- Calculer la longueur d'un segment
- Translater un point
- Vérifier si deux points sont égaux
- Afficher un point sous forme d'une chaîne de caractères

Voici la spécification (incomplète) des opérations sur cette structure de données :

- `Point(x, y)` : crée un élément de type Point à partir de deux flottants `x` (abscisse) et `y` (ordonnée).
- `abscisse(p)` : accès à l'abscisse du point `p` (renvoie un flottant).
- `modifier(p, x, y)` : modifie les coordonnées d'un point `p` déjà créé en les remplaçant par `x` (nouvelle abscisse) et `y` (nouvelle ordonnée).
- `milieu(p1, p2)` : renvoie un nouveau point correspondant au milieu du segment d'extrémités `p1` et `p2`.
- `longueur(p1, p2)` : renvoie la longueur du segment d'extrémités `p1` et `p2`.
- `translater(p, dx, dy)` : modifie les coordonnées d'un point `p` déjà créé pour qu'elles correspondent à la translation du vecteur de coordonnées `(dx, dy)`.
- `egal(p1, p2)` : renvoie `Vrai` si les deux points `p1` et `p2` sont égaux, `Faux` sinon.

## PARTIE 1

1. Complétez l'interface de manière à spécifier les opérations manquantes.
2. Quelle instruction permet de créer le point $M(-3, 1)$ ?
3. Après avoir exécuté le programme suivant, quelle est la valeur renvoyée par `egal(C, D)` ?

```
A = Point(1, 2)
B = Point(-3, 5)
C = Point(3, 1)
D = Point(-abscisse(B), abscisse(A))
```

4. Quelle est la valeur renvoyée par `longueur(A, B)` ?
5. On souhaite créer le point $I$ milieu du segment $[AC]$ puis lui appliquer une translation de vecteur $(3, -2)$ et enfin afficher ses coordonnées. Ecrivez les instructions nécessaires. Quelle chaîne de caractères doit s'afficher à l'écran ?

## PARTIE 2

L'objectif est d'écrire deux implémentations de la structure de données `Point` définie dans la partie1. On utilisera un **dictionnaire** dans la première et un **tableau** dans la seconde.

### Implémentation avec un dictionnaire

On choisit ici d'utiliser un dictionnaire pour représenter le type de données abstrait `Point`.

1. Complétez l'implémentation qui suit pour programmer toutes les opérations définies sur la structure de données Point. Vous préciserez dans la docstring des différentes fonctions, les types des arguments et des éventuelles valeurs renvoyées.

```
# IMPLEMENTATION AVEC UN DICTIONNAIRE
def Point(x, y):
        """Flottant x Flottant --> Point"""
        return {"abs": x, "ord": y}

def abscisse(p):
        """Point --> float"""
        # A COMPLETER

def ordonnee(p):
        # A COMPLETER

def modifier(p, x, y):
        """Point x Flottant x Flottant --> Rien"""
        p["abs"] = x
        p["ord"] = y

def milieu(p1, p2):
        # A COMPLETER

def longueur(p1, p2):
        """Point x Point --> Flottant"""
        # A COMPLETER

def translater(p, dx, dy):
        # A COMPLETER

def egal(p1, p2):
        # A COMPLETER

def afficher(p):
        """Point --> Rien
        Affiche les coordonnees de p sous forme d'une chaine de caracteres."""
        print("(" + str(p["abs"]) + ";" + str(p["ord"]) + ")")
```

2. Vérifiez que l'implémentation est correcte en testant différentes instructions (par exemple, celles de la partie 1)

### Implémentation avec un tableau

Dans cette seconde implémentation, vous allez représenter un point par un tableau de taille 2.

1. Au regard des opérations sur la structure de données `Point`, expliquez pourquoi il n'était pas possible de représenter les points par des tuples (de taille 2).
   **Important** : le programmeur qui implémente une structure de données abstraite doit représenter le type abstrait avec des types existants dans le langage et qui sont compatibles avec l'interface communiquée.
   Le choix de la représentation est donc guidé par l'interface de la structure de données.
2. Proposez une implémentation de la structure de données abstraite `Point` en utilisant un tableau.
3. Vérifiez que l'implémentation est correcte en testant différentes instructions. Comparez les deux implémentations.



Source : Lycée Mounier - Angers