TP04 – Collections & Sérialisation

Objectifs du TP

- Manipulation des collections & itérateur
- Utiliser la sérialisation JSON
- Utilisation d'une API « RESTful »

Exercice 1 – Premier Exercice

1 – Préparation avant la séance de TP

Faire le diagramme de classes pour le contenu du dossier « core »

2 – Réalisation en séance

Ajouter le code nécessaire (identifié par des commentaires To-Do) dans les classes :

- `LedActuator` : pour agir sur la bonne LED
- `TemperatureSensor` : pour récupérer la valeur du capteur de température depuis la SenseHat
- `HumiditySensor` : pour récupérer la valeur du capteur d'humidité depuis la SenseHat
- `PressureSensor` : pour récupérer la valeur du capteur de pression depuis la SenseHat

Lancement du serveur sur la carte RPi :

- Ouvrir un terminal sur la cible (votre carte RPIi+SenseHat)
- Se positionner dans le dossier contenant le fichier `sense_server.py`

```
cd "/chemin/absolu/vers/le/dossier/contenant/le/fichier_sense_server_py"
```

- Lancer le serveur :

```
python sense_server.py
```

Test du serveur :

- Depuis un navigateur, tester les URLs[1] suivantes:

  http://<adresse_ip_de_votre_carte>:8888/devs

  http://<adresse_ip_de_votre_carte>:8888/devs/sensors

  http://<adresse_ip_de_votre_carte>:8888/devs/actuators

  http://<adresse_ip_de_votre_carte>:8888/devs/sensors/temp

  http://<adresse_ip_de_votre_carte>:8888/devs/sensors/temp/last

  http://<adresse_ip_de_votre_carte>:8888/devs/sensors/temp/history

  http://<adresse_ip_de_votre_carte>:8888/devs/actuators/L00/last

  http://<adresse_ip_de_votre_carte>:8888/devs/actuators/L00/action/on/255/0/0

  http://<adresse_ip_de_votre_carte>:8888/devs/actuators/L00/action/on/0/0/255

  http://<adresse_ip_de_votre_carte>:8888/devs/actuators/L00/action/off

Faire valider

Exercice 2 – Deuxième Exercice

1 : URL = Uniform Resource Location

Le but de cet exercice est d'écrire un programme Python (client) s'exécutant sur votre PC et qui interagit avec le serveur qui s'exécute sur votre carte RPi et qui pilote le shield SenseHat. Les communications utilisent le protocole http et le format des données échangées est JSON.

1 – Réalisation en séance

Soit le code suivant d'un client REST en Python

```python
import json
import requests

#ToDo changer l'adresse IP pour correspondre à celle de votre carte RPi
ip_address = '127.0.0.1'
port_num = 8888
url = f'http://{ip_address}:{port_num}/devs/sensors/temp/last'
headers = {'Accept': 'application/json'}

# Exécution de la requête http spécifiée par son URL
response = requests.get(url, headers=headers)

# Récupération de la réponse sous la forme JSON
jsonObject = response.json()

print(json.dumps(jsonObject, indent=4))
```

- Depuis une nouvelle fenêtre VSCode (non connecté à votre carte) :
  - Dans un terminal, installer le package `requests` à l'aide de la commande suivante :

    ```
    pip install requests
    ```
  - Exécuter et tester ce programme.
  - Tester les autres URLs de l'exercice précédent.

Faire valider

Exercice 3 – Troisième Exercice

Le but de cet exercice est de structurer le programme Client selon le paradigme de la POO.

1 – Réalisation en séance

Définir une classe `RemoteServer` qui a pour :

- Attributs (initialisés à l'instanciation):

1

1 : URL = Uniform Resource Location

  - `ip` : Adresse IP de la carte RPi+SenseHat
  - `port` : Numéro de port d'écoute du serveur tournant sur la carte.
  - `baseURL` : partie servant de base à toutes les futures URLs :

    `http://<adresse_ip>:{num_port}`

- Méthodes :
  - `doRequest(path)` : qui exécute une requête http et retourne l'objet JSON contenu dans la réponse. La paramètre `path` contient ce qu'il y a après `baseURL`.

Définir une classe `RemoteDevice` qui a pour :

- Attributs :
  - `name` : Nom du device (initialisé à l'instanciation)
  - `path` : chemin relatif pour l'accès à ce device initialisée à None
  - server : une instance de RemoteServer initialisé à None. Cet attribut sera initialisé à l'utilisation (cf. méthode setServer)
- Méthodes
  - `setServer(server)` : associe un serveur au device
  - `getRepresentation()` : méthode abstraite

Définir une classe `RemoteSensor` héritant de `RemoteDevice` et qui a pour :

- Méthodes
  - Un constructeur redéfini pour prendre en charge la mise à jour de l'attribut hérité `path`.
  - `getRepresentation()` : méthode qui récupère la description du capteur auprès du serveur.
  - `getLastValue()` : méthode qui renvoie la dernière valeur du capteur depuis le serveur.
  - `getHistory()` : méthode qui renvoie la dernière l'historique des valeurs du capteur depuis le serveur.

Définir une classe `RemoteLed` héritant de `RemoteDevice` et qui a pour :

- Méthodes
  - Un constructeur redéfini pour prendre en charge la mise à jour de l'attribut hérité `path`.

1 : URL = Uniform Resource Location

- `getRepresentation()` : méthode qui récupère la description du capteur auprès du serveur.
- `switchOn(r, g, b)` : méthode qui allume la led distante avec la couleur spécifiée
- `switchOff()` : méthode qui éteint la led distante

Programme principal :

- Récupérer auprès du serveur le nom de tous les devices exposés par le serveur et les organisées dans deux listes distinctes : `sensorNameList` et `ledNameList`.
- Créer une liste d'instances de `RemoteSensor` : `sensors` pour chaque capteur dont le nom est dans la liste `sensorNameList`
- Créer une liste d'instances de `RemoteLed` : LEDs pour chaque led dont le nom est dans la liste `ledsNameList`
- Dans une boucle infinie :
  - Récupérer la dernière valeur de chaque capteur et l'afficher à l'écran
  - Chaque 10 itérations :
    - Afficher l'historique récupéré auprès du serveur pour chaque capteur
  - Choisir une led au hasard et lui affecter une couleur aléatoire
  - Attendre un délai de 2 secondes

1 : URL = Uniform Resource Location