# **TP2 :**

# **Mise en contexte :**

On commence à mettre en place un système de gestion des sous-réseaux locaux.

Nous voulons une classe gestionnaire qui permet de gérer 1 ou plusieurs sous-réseaux. Ces sous-réseaux son composé d’instances de la classe composante, des postes de travail ou bien des routeurs.

Chaque composante contient et possède une adresse IP. Pour gérer la validité des adresses IP, nous allons créer une classe AdresseIP. Pour simplifier la situation, les valeurs de masques sous-réseau seront aussi considérées comme étant des adresses IP.

Une image contenant texte, capture d’écran, logiciel, conception

Description générée automatiquement

Dans ce TP, vous aurez 7 classes à modéliser.

1. Dans le fichier **AdresseIP.py** vous allez créer les classes ErreurFormatIP et AdresseIP :

La classe **ErreurFormatIP** hérite de ValueError et est utilisé dans la classe AdresseIP

La classe AdresseIP contient un attribut privé correspondant à un str de l’adresse ip. La classe adresse ip possède une propriété pour accédé à cet attribut et un setter pour changé l’attribut. LE constructeur et le setter font tous deux appel à la méthode vérifier\_format\_adresse() afin d’être sur qu’il s’agit bien d’une adresse IP. La classe redéfini aussi deux méthodes « magiques » : \_\_str\_\_ pour pouvoir imprimer la classe et la convertir en string, et \_\_eq\_\_ pour pouvoir comparer deux adresses ip.

1. Dans le fichier **Composante.py** vous allez créer les classes Composante, Poste, et Routeur :

La classe abstraite **Composante** a des attributs privés et un attribut public. Le nom sera en lecture seulement, mais les autres attributs auront des accesseurs pour qu’on puisse les modifier. Les classes dérivées devront implémenter la méthode abstraite fermer().

La classe **Poste** (un poste de travail) hérite de Composante. Elle a 3 méthodes.

La classe **Routeur** est une classe qui hérite de Composante. Elle a une liste de connexions. Elle peut ajouter des connexions jusqu’à ce que son nombre de ports soit atteint. Elle gère la connexion et la déconnexion d’une station à la fois.

1. Dans le fichier **TP2\_Gestionnaire.py** vous allez créer les classes SousReseau et Gestionnaire.

La classe **SousReseau** a une liste de composantes. Elle a une méthode qui permettra d’obtenir une composante du réseau par son adresse\_ip. Elle a aussi une méthode ping() qui activera le ping() de la composante si elle n’est pas un commutateur. Enfin elle a deux méthodes pour ajouter et enlever des composantes au réseau.

Vous avez la classe **Gestionnaire** qui fera la gestion d’un réseau. On a évidemment simplifié les actions à faire pour faire une bonne gestion d’un réseau autrement le TP serait trop long. Elle a une méthode pour tester la connexion d’une composante, une méthode pour redémarrer un poste de travail et une méthode pour tester toutes les connexions du sous-réseau.

# **Création des différentes classes dans Python.**

Regardez le diagramme UML, ajoutez les classes manquantes et complétez la signature des méthodes des nouvelles classes et de celles qu’on a déjà commencées en utilisant **pass** pour terminer chaque méthode tout de suite.

**Dans GitHub, faites une remise à cette étape des deux fichiers, avec votre NOM et PRÉNOM dans le haut de chacun des fichiers.**

**Appelez cette remise « FCT : Création des signatures des classes »**

# **Complétez les différentes classes dans Python en vous référant au diagramme UML et aux tableaux qui suivent.**

**Chacun des tableaux qui suit décrit chaque classe plus en détail, avec ses attributs, propriétés et méthodes.**

**Dans GitHub, faites une remise après avoir terminé le code de CHAQUE classe.**

**Dans la remise, mentionnez le nom de la classe que vous venez de compléter et le préfixe FCT.**

Par exemple, après avoir complété la classe composante, vous faites une remise de **TP2\_Composantes.py** avec le texte : **FCT Classe Composante.**

1. **Dans AdresseIP.py**

|  |
| --- |
| Classe **E**rreurFormatIP  Cette classe hérite de la classe ValueError. Elle sera soulevé dans la classe AdresseIP lorsque la valeur d’adresse passé en paramètre n’est pas |

**Dans GitHub, faites une remise après avoir terminé le code de CHAQUE classe.**

|  |  |
| --- | --- |
| Classe **AdresseIP**  C’est une classe **abstraite**. | |
| **Attributs** | |
| \_valeur | Attribut privé. Un string. |
| **Propriétés** |  |
| valeur | Donner accès en lecture et en écriture  Le setter va prend un str en paramètre et utilisera la méthode **verifier\_format\_adresse()** pour s’assurer qu’il s’agit bien d’une adresse IP |
| **Méthodes** |  |
| \_\_init\_\_() | **Constructeur**.  Prend un str en paramètre correspondant a une adresse IP. Vérifie qu’il s’agit bien d’une adresse IP en utilisant la méthode **verifier\_format\_adresse()** |
| verifier\_format\_adresse() | Méthode **statique** qui prend en paramètre un str..  Vérifier que l’adresse est constituée de 4 portions et que chacune des portions de l’adresse est un nombre entier entre 0 et 255. Si ce n’est pas le cas, soulever une erreur de type **ErreurFormatIP** avec un message comme quoi le format adresse IP est non valide. |
| \_\_str\_\_() | Retourne un str correspondant à l’attribut \_valeur de l’instance |
| \_\_eq\_\_() | Prend un objet de type AdresseIP en paramètre en plus du self.  Retourne **True** si la propriété valeur des deux objets est la même. Retourne **False** dans le cas contraire. |

1. **Dans Composantes.py**

|  |  |
| --- | --- |
| Classe **Composante**  C’est une classe **abstraite**. | |
| **Attributs** | |
| \_nom | Attribut privé. |
| \_adresse\_ip | Attribut privé. Instance de la classe AdresseIP |
| \_masque\_sous\_reseau | Attribut privé. Instance de la classe AdresseIP |
| \_adresse\_passerelle | Attribut privé. Instance de la classe AdresseIP |
| location\_physique | Attribut public qui enregistrera l’endroit où la composante est située physiquement. |
| **Propriétés** | |
| nom | Donner accès en lecture seulement |
| adresse\_ip | Donner accès en lecture et en écriture  Le setter va prend un str en paramètre, crée une instance de la classe AdresseIP et changer la valeur de \_adresse\_ip |
| masque\_sous\_reseau | Donner accès en lecture et en écriture  Le setter va prend un str en paramètre, crée une instance de la classe AdresseIP |
| adresse\_passerelle | Donner accès en lecture et en écriture  Le setter va prend un str en paramètre, crée une instance de la classe AdresseIP |
| **Méthodes** | |
| \_\_init\_\_() | **Constructeur**.  Reçoit un 5 str en paramètre.  Crée 3 nouvelles instances de la classe AdresseIP |
| ping() | Le code est déjà écrit parce qu’il utilise le module subprocess qui a été vu il y a pas mal de temps déjà.  Il fait le ping de cette composante.  La méthode retourne **True** quand la composante répond au ping et, **False** sinon. |
| *fermer*() | Méthode abstraite.  Le code de cette méthode devra être fait dans les classes dérivées. |

**Dans GitHub, faites une remise après avoir terminé le code de CHAQUE classe.**

|  |  |
| --- | --- |
| Classe **Poste**  Classe dérivée de la classe Composante. | |
| **Attributs** | |
| est\_allume | Attribut public qui prendra la valeur True quand la station est ouverte, False sinon.  Au départ, la station sera fermée. |
| **Propriétés** | |
| **Méthodes** | |
| ouvrir() | Si la station est déjà allumée, vous devez soulever une exception.  Sinon, l’attribut **est\_allume** devient True .  (Dans la mise en situation nous sommes en train de COMMENCER le système de gestion. D’autre fonctionnalité serait ajoutée plus tard dans le développement) |
| fermer() | Si la station est déjà fermée, vous devez soulever une exception.  Sinon, l’attribut **est\_allume** devient False.  (Dans la mise en situation nous sommes en train de COMMENCER le système de gestion. D’autre fonctionnalité serait ajoutée plus tard dans le développement) |

|  |  |
| --- | --- |
| Classe **Routeur**  Classe dérivée de la classe Composante. | |
| **Attributs** | |
| \_nb\_ports | Attribut privé. Pour savoir le nombre de ports que le routeur possède. |
| \_ls\_connexions | Attribut privé qui enregistrera les connexions faites dans ce routeur. |
| **Propriétés** | |
| nb\_port | Donner accès en lecture seulement |
| ls\_connexions | Donner accès en lecture seulement |
| **Méthodes** | |
| ajouter\_connexion() | Prends en paramètre une instance de la classe Composante.   1. La méthode ajoute la composante dans la liste des connexions du routeur. 2. Avant d’ajouter la composante à la liste de connexions du routeur, il faut vérifier qu’on n’a pas déjà atteint le nombre maximal de connexions possibles pour ce routeur. 3. **SI** la composante ajoutée est un routeur, il faut ajouter le routeur courant (le self) à ce routeur ci. (Un peu plus difficile, garder cela pour la fin. La fonction isinstance() peut être utile dans cette métode ) |
| supprimer\_connexion() | Prends en paramètre une instance de la classe Composante.  Méthode qui enlève la composante de la liste des connexions du routeur. |
| fermer() | Le code est déjà fait pour vous.  (Dans la mise en situation nous sommes en train de COMMENCER le système de gestion. D’autre fonctionnalité serait ajoutée plus tard dans le développement) |

1. **Dans TP2\_gestionnaire.py**

|  |  |
| --- | --- |
| Classe **SousReseau** | |
| **Attributs** | |
| \_nom | Attribut privé. |
| \_ls\_composantes | Attribut public qui enregistrera les composantes du réseau. |
| **Propriétés** | |
| nom | Donner accès en lecture seulement |
| ls\_composantes | Donner accès en lecture seulement |
| **Méthodes** | |
| ping() | Cette méthode prend en paramètre une composante puis retourne le résultat de la méthode ping() de la cette composante. |
| get\_composante\_par\_ip() | Prends en paramètre un str correspondant à une adresse ip.  La méthode passe à travers sa liste des composantes. Si l’adresse\_ip de la composante courante est égale à l’adresse passé en paramètre, elle retourne la composante.  Autrement, elle retourne None. |
| ajouter\_composante() | Prends en paramètre une composante.  Vérifie si la composante fait déjà partie de \_ls\_composantes et ajoute la composante si ce n’est pas le cas. |
| supprimer\_composante() | Prends en paramètre une composante.  Méthode qui enlève la composante de la liste des composantes du réseau. |

**Dans GitHub, faites une remise après avoir terminé le code de CHAQUE classe.**

|  |  |
| --- | --- |
| Classe **Gestionnaire** | |
| **Attributs** | |
| \_nom | Attribut privé. Vous lui donnerez la valeur de votre nom quand vous instancierez un objet Gestionnaire. |
| \_reseau | Attribut privé. Correspond à UNE instance de la classe SousReseau |
| **Propriétés** | |
| nom | Donner accès en lecture seulement |
| reseau | Donner accès en lecture seulement |
| **Méthodes** | |
| tester\_connexion() | Cette méthode prend en paramètre une composante  Vous capturez dans une variable le résultat de la méthode ping() de la composante du réseau .  Le résultat du ping sera True ou False. Imprimez un message avec le nom de la composante et le résultat du test. |
| redemarrer\_station() | Méthode **statique**. Elle prend en paramètre une instance de la classe **Poste**.  Vous appelez la méthode **fermer()** de la station, attendez 1 seconde (avec **time.sleep(1))** et vous appelez enfin la méthode **ouvrir()** de la station. |
| tester\_toutes\_les\_connexions() | Méthode d’instance. Elle n’a pas d’autre paramètre que self. Passe à travers toutes les composantes de tous les réseaux dans \_ls\_reseau et appelle la méthode **tester\_connexion()** pour chaque composante. |

Pour les messages, vous trouverez à la page suivante un exemple de résultat que vous pourriez avoir eu pour les tests qui sont un succès.

# **Instanciez des objets des différentes classes et testez le code.**

Vous allez compléter 2 fichiers python :

* **TP2\_tests\_succes.py** pour tester ce qui devrait fonctionner.
* **TP2\_tests\_echec.py** pour tester les données hors limites qui ne devraient pas fonctionner.

**Chacun des fichiers a présentement des commentaires pour vous aider : les instances des objets à réaliser ainsi que les tests à faire et le résultat attendu dans le terminal.**

## **TP2\_tests\_succes.py**

Commencez par faire une série de tests qui doivent fonctionner, en suivant les indications données par les commentaires dans le script.

**AU FUR ET À MESURE QUE VOUS FAITES VOS TESTS, VOUS ALLEZ CORRIGER DES BUGS DANS VOS CLASSES.**

**Vous ferez une remise de la nouvelle version du code, avec BUG\_FIX comme préfix.**

Par exemple, après avoir trouvé et corrigé des erreurs dans la classe composante, vous faites une remise de **TP2\_Composantes.py** avec le texte : **BUG FIX Classe Composante**

Vous allez probablement avoir des erreurs. N’hésitez pas à utiliser le débogueur pour voir la progression de votre code pas à pas.  
  
**Quand toute votre série de tests fonctionne, faites une remise de TP2\_tests\_succes.py dans GitHub. FCT : Tests succès**

**Avec probablement quelques BUG\_FIX pour les classes qui auront été modifiées à la suite des tests.**

Une fois que tout est fini, vous devriez afficher les succès des tests dans un format similaire à celui qui suit :

### Résultat attendu des tests à succès :

Tests 1, instanciation d'objets

Tests 2, Postes :

T2a - SUCCÈS - La station ORDI\_01 est éteinte.

T2b - SUCCÈS - La station ORDI\_01 est allumé.

T2c - SUCCÈS - La station ORDI\_01 est éteinte.

T2d - SUCCÈS - La station ORDI\_01 répond aux pings.

Tests 3, routeurs :

T3a - SUCCÈS - Les stations ORDI\_01 et ORDI\_02 ont été ajouter avec succès.

T3b - SUCCÈS - La station ORDI\_01 à été retiré avec succès.

T3c - Le sous-réseau routeur1 contient les connexions suivantes :

ORDI\_02

T3d - SUCCÈS - Le router routeur1 répond aux pings.

T3e - SUCÈS - Le routeur 1 fait bien partie de la liste de connexions de routeur 2

Tests 4, réseau :

T4a - SUCCÈS - Les composantes ORDI\_01,ORDI\_02,, et routeur1 ont bien été ajoutés.

T4b - SUCCÈS - La station ORDI\_01 à été retiré avec succès.

T4c - SUCCÈS - Le réseau reseau INFO est capable de demander à routeur1 de répondre aux pings.

T4d - SUCCÈS - Le réseau est capable d'aller chercher ORDI\_01 à partir de son adresse IP.

T5e - Le réseau reseau INFO contient les composantes suivantes :

ORDI\_01

ORDI\_02

routeur1

Tests 5, gestionnaire Gestionnaire\_gallant\_pierre\_paul :

T5a - ORDI\_01 208.67.222.222 répond ? True

T5b - ORDI\_02 255.67.223.223 répond ? False

T5c - Test des connexions :

ORDI\_01 208.67.222.222 répond ? True

ORDI\_02 255.67.223.223 répond ? False

routeur1 8.8.8.8 répond ? True

## **TP2\_tests\_echec.py**

Complétez le script pour la série de tests qui vérifie les valeurs limites. Ces tests devraient tous soulevé une exception. Suivez les commentaires qui ont été mis dans le script.

Vous devrez mettre chaque test dans un bloc try…except qui vérifie que la bonne erreur a été soulevé

Vous allez probablement trouver des erreurs qui ne sont pas attendu ou qui ne soulève pas d’exceptions. N’hésitez pas à utiliser le débogueur pour voir la progression de votre code pas à pas.  
  
**Quand toute votre série de tests qui ne fonctionnent pas ont été complétés et testés, faites une remise de TP2\_tests\_echec.py dans GitHub : FCT Tests en échec.**

**Avec probablement quelques BUG\_FIX pour les classes qui auront été modifiées à la suite des tests.**

# **Félicitation. Vous avez complété votre TP2!**

**À la toute fin, compressez votre dossier TP2 et faites une remise dans LÉA.**

# **Grille de correction du TP2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Création et remise de la signature des classes dans GitHub** | | | **15** |
|  | Création initiale du dépôt | | 3 |
|  | Création de la signature au début | | 2 |
|  | Messages commits significatifs | | 10 |
|  |  | |  |
| **classe AdresseIP** | | | **12** |
|  | Méthode vérifier\_format\_adressse | | **5** |
|  | Vérifie le format dans constructeur et dans setter | | **5** |
|  | Soulève l'exception InvalidFormatIP | | **2** |
|  |  | |  |
| **Classe composante** | | | **14** |
|  | | Constructeur (attributs privés) | 2 |
|  | | Instanciation d'objet AdresseIp | 5 |
|  | | Classe & méthode abstraite | 3 |
|  | | Propriétés (getter & setters) | 2 |
|  | | Setter de l'adresseip | 2 |
|  | |  |  |
| Classe station | | | **4** |
|  | Héritage | | 3 |
|  | Inialisé est\_allumer = False | | 1 |
|  |  | |  |
| Routeur | | | **15** |
|  | nb\_ports (privé + propriété) | | 2 |
|  | ajouter\_connexion() | | 4 |
|  | capable d'ajouter un routeur de la bonne façon | | 5 |
|  | supprimer\_connexion | | 4 |
|  |  | |  |
| SousReseau | | | **12** |
|  | ping | | 3 |
|  | composante par ip | | 3 |
|  | ajouter composante | | 3 |
|  | enlever composante | | 3 |
|  |  | |  |
| Gestionnaire | | | **13** |
|  | constructeur (init et propriété nom) | | 3 |
|  | tester\_connection | | 4 |
|  | tester\_toutes \_connections | | 4 |
|  | redemarrer\_station | | 2 |
|  |  | |  |
| Tests fonctionnels (TP2\_tests\_succes.py) | | | **15** |
| Tests limits (TP2\_tests\_echec.py) | | | **15** |
|  |  | |  |
|  | Total : | | **130** |
| Pts enlevés si vous n’avez aucune remise qui est un BugFix dans GitHub | | | 5 |
| Pts enlevés si vos noms de variables ne sont pas significatifs | | | 10 |