TP 2: Data Access Objet (DAO) et Web Service

Dans ce TP vous allez :

- Implémenter le patron de conception DAO
- Manipuler un webservice
- Voir si votre programme fonctionne avec des tests unitaires reproductibles

1 Introduction

Dans ce TP nous allons mettre en place la **persistance** de nos données et commencer à manipuler un **webservice**.

> 🤔Pourquoi persister nos données ?

Actuellement si rien n'est fait lorsque notre application s'éteint, toutes les objets qu'elle manipulait sont perdus. Cela est du au fait que python travaille avec la *mémoire vive* de votre ordinateur (mémoire RAM pour Random Acess Memory). C'est une mémoire très rapide mais volatile, avec une place limitée (plus que la mémoire de votre disque dur). Volatile signifie que quand votre ordinateur éteint, vos barrettes de RAM (l'objet physique qui contient la mémoire) ne sont plus alimentées, et toutes les informations qu'elles contenaient sont perdues. L'autre problème provient de sa gestion par votre ordinateur. Pour faire simple, quand un programme tourne et crée des objet en RAM, lorsque ce programme s'éteint les objets en mémoire RAM sont perdus.

Il nous faut donc un moyen de persistez nos données. Pour ce la il y a deux grandes solutions :

- **Sérialiser** les données pour les écrire dans un fichier, qui sera stocker sur le disque dur de la machine. On pourrait par exemple les écrire dans un fichier json

```
{
    "nom" : "Link",
    "force" : 10,
    "agilite" : 20,
    "magie" : 5,
    "defense" : 13,
    "point de vie": 55,
    "arme" : {
        "nom" : "master sword",
        "phrase_attaque" :

    }
}
```

  Il nous suffirait ensuite de lire ce fichier pour recréer le personnage.

- Le sauvegarder dans une base de données (SQL ou no-SQL).

Si les deux méthodes sont utilisables, il faut bien avoir conscience de leurs différences :

- La base de données fourni des outils de requêtage et administration des données. Pour le requêtage vous connaissez le langage SQL (Structured Query Language) pour les bases de données relationnelle, mais d'autres langage existent pour d'autres bases. Le SQL est un langage haut niveau pour manipuler des données. Même s'il est normalisé, tous les systèmes de gestion de bases de données (SGBD) ont leur petites particularité. L'inconvénient de l'utilisation d'une base de données et qu'il faut avoir une base de données à disposition.
- La gestion de la persistance par sérialisation suppose de redévelopper tout un système de gestion des données, ou d'utiliser une API (*Application Programming Interface*) qui va vous aider à manipuler vos données (dans ce cas là on tombe sur une solution assez proche de la base de données avec un outil qui intègre déjà tout ce qu'il faut pour manipuler les données).

Pour résumer :

- **Base de données** : propose une interface de gestion des données intégrée mais demande l'installation d'une base
- **Sérialisation** : développer ses propres outils, ou utiliser des API qui les proposent.

Pour rappel voici les 4 opérations de base sur les données (et leur équivalent en SQL) :

- **C**reate (Insert)
- **R**ead (Select)
- **U**pdate (Update)
- **D**elete (Delete)

> 🤔Souvent des élèves se demandent pourquoi l'opération de copie ne fait pas partie des opérations de base. En fait copier une donnée revient à la lire (read) et l'écrire (create). Par exemple copier une ligne en SQL s'écrit :
>
> ```
> 1 INSERT INTO table SELECT * FROM table where condition
> ```

2 Mise en place

- Ouvrez deux terminaux vers le cluster de calcul en utilisant Putty. Un servira pour git, et un autre pour pycharm. En cas de difficulté référez-vous au document sur moodle pour la marche à suivre.
- Dans le terminal pour git, retournez dans le dossier où vous avez téléchargé le code du premier TP en naviguant dans le terminal avec des `cd` puis fait un `git checkout`. Si vous avez respecté les instructions du TP1 il vous suffit de faire.

```
1 cd 2A/complement_info_ensai_2020_2021
2 git checkout TP2_ex1
```

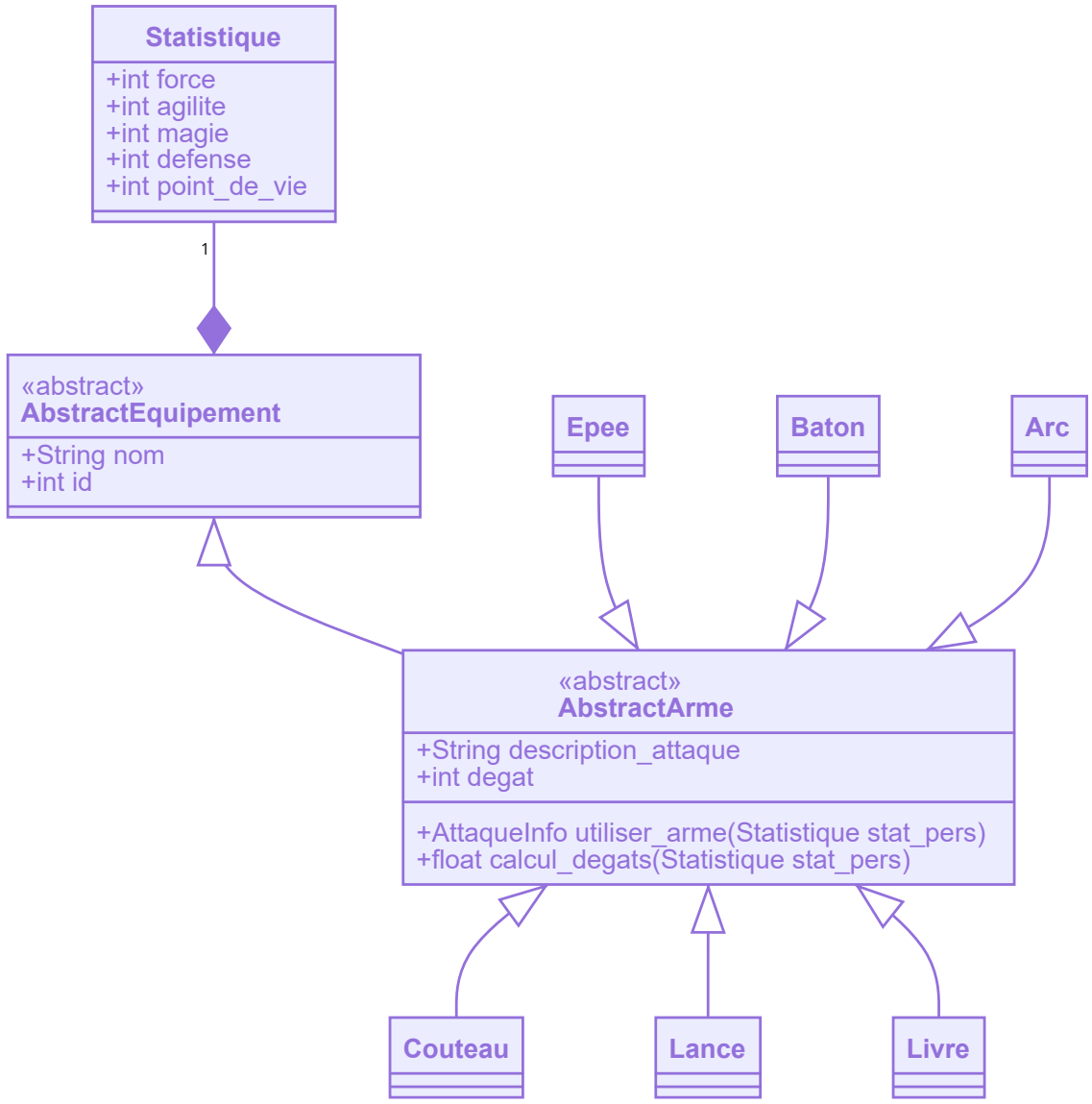
- Dans le terminal pour Pycharm, lancez Pycharm en tapant `pycharm-community`
- Ouvrez le le dossier contenant le code du TP
- Vérifiez que tous est bien configurer et lancez les testes du projet avec un clic droit sur le package test puis "Run Unittests in test". Normalement seul 3 tests devraient être en erreur, tous dans la classe test_armeDAO
- Dans le fichier propreties dans le package configuration fixez les valeurs suivantes :
    - host = "sgbd-eleves.domensai.ecole"

- port = "5432"
- database = "votre idep"
- user = "votre idep"
- password = "votre idep"

3 Data Acess Objet (DAO)

Modélisation

Reprenons le diagramme de classe du TP précédent et limitons nous à la partie "arme" (pas les personnage est armure) et réfléchissons où mettre une méthode qui permet de persister les armes



> 🔍 Les plus observateurs d'entre vous auront remarqué l'apparition d'un attribut id dans la classe AbstractEquipement. Ce n'est pas le seul changement comparé à la semaine dernière. Ils ne changent pas le fonctionnement général du code, mais sont là pour éviter la redondance de code, et permettent des tests plus simples.

Bon déjà vu que les attributs de nos armes sont similaires on ne va pas coder cela dans les classes des spécifique des armes. On pourrait mettre les méthodes dans AbstractArme. Et cela fonctionnerait. Mais on ne va pas faire ça !

Et là vous êtes en droit de vous demander

> 😱 Mais pourquoi ???

Et la réponse est

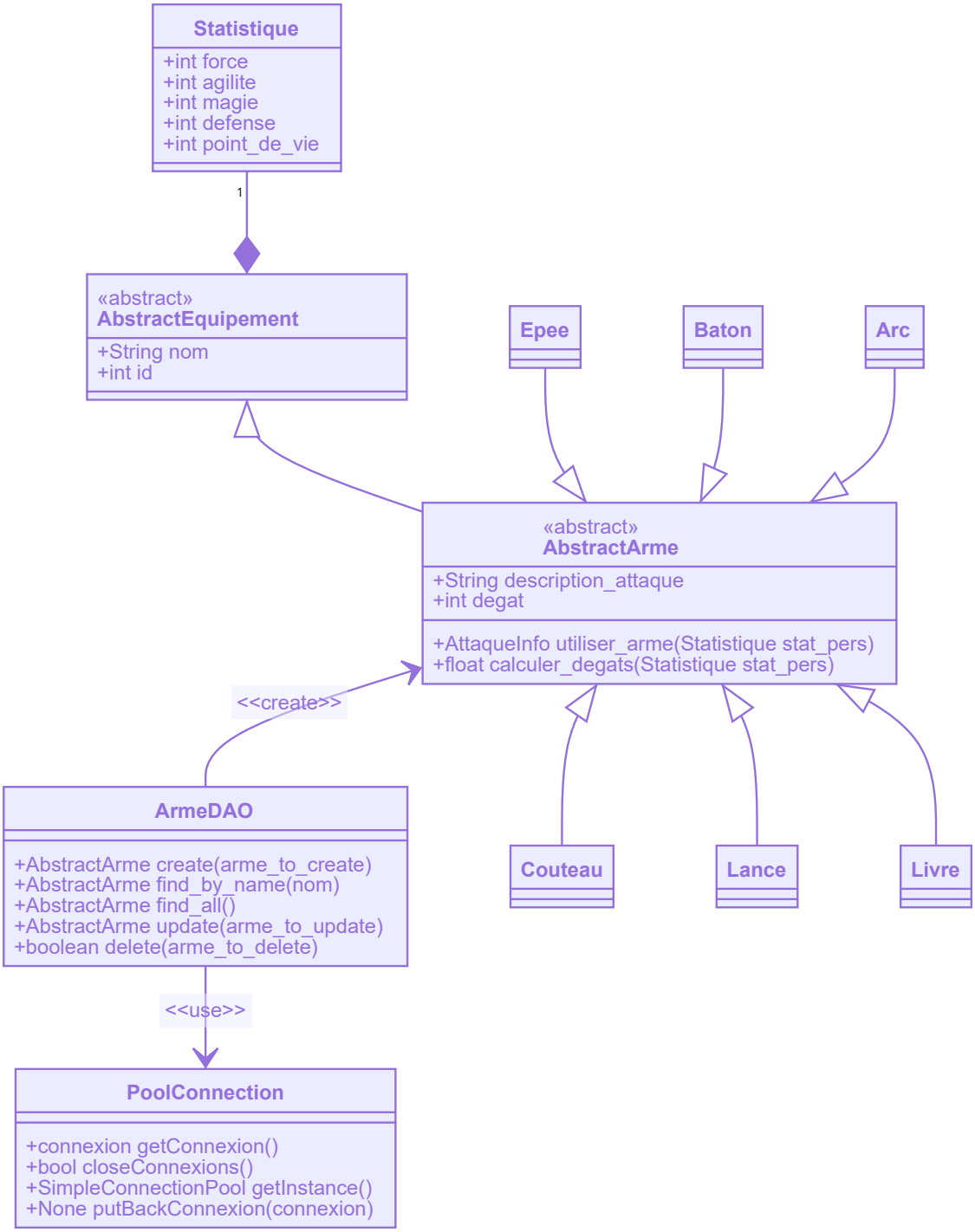
> 😛 Car ça n'a aucun sens

Revenons sur la phrase : **faible couplage, forte cohésion**. Si on met toutes les méthodes de persistance de nos armes dans la classe AbstractArme on va avoir une classe qui :

- ✔ Détermine le comportement des armes.
- ✕ Détermine comment on persiste une arme.

Mais ça, ce n'est pas de la responsabilité d'une arme, mais du système de persistance choisi ! Et je n'ai personnellement pas envie d'aller modifier ma classe AbstractArme uniquement car j'ai décidé de changer de système de gestion de la persistance. Je risque de modifier quelque chose que je ne devrais pas et créer des régressions dans mon code. Or j'aimerais bien limiter les sources d'erreur

À la place on va créer une classe qui va s'occuper uniquement de cette tâche. Et on appelle ce type de classe DAO pour **Data Acess Object**. C'est une classe technique qui va faire l'interface entre mes données stockées et mon application. Voilà ce que cela donne en terme de diagramme de classe



> 😅 Cela commence à faire beaucoup de classe et ce n'est qu'un début. Je ne cache pas qu'on va terminer avec pas mal de classes. Mais toutes nos classes auront une tâche bien précise. Cela aide à la lisibilité du code, ça maintenabilité et son débuggage. Par exemple la classe ReservoirConnexion sert uniquement à gérer les connexions avec la base. S'il y a un problème avec les connexion je sais le problème vient de cette classe. Et cela évite d'avoir des bout de code pour gérer les connexions partout.

Gestion des connexions et patern singleton

Pour nous connecter à la base de données nous allons utiliser la bibliothèque python `psycopg2`. C'est elle qui va établir la connexion avec la base, envoyer nos requêtes et nous retourner les résultats. Mais il faut faire un peu attention à la gestion des connexions. Car on pourrait se retrouver à ouvrir des centaines de connexion rapidement et dégrader les performances de notre application. C'est le travail de la classe `PoolConnection`.

La classe `PoolConnection` est une classe singleton (singleton patern), c'est à dire que toutes les instances de la classe `PoolConnection` pointent vers le même objet. La première fois qu'une connexion sera demandée, un *pool* (réservoir) de connexion sera initialisé pour l'application. Et à partir de là chaque fois qu'une connexion sera requise, une connexion de se *pool* sera utilisée. Comme ça, on s'assure d'utiliser un nombre constant de connexion dans notre application.

Cette classe est une solution purement technique alors n'hésitez pas à la réutiliser pour votre projet. Le code de la classe est bien documenté pour les personnes intéressées.

Si vous voulez plus d'information voici un tuto en anglais : lien

DAO et CRUD

Si vous faites attention, les méthodes de notre DAO ressemblent à celles du CRUD. C'est normal car c'est dans ces méthodes que le code SQL va être stocké, donc il nous faut les méthodes de bases. Néanmoins pour gagner du temps rien n'empêche de créer des méthodes plus complexe. Par exemple il y a deux méthodes pour lire des données :

- find_by_id : qui retourne juste l'enregistrement avec l'id souhaité ;
- find_all : qui va retourner toute une table.

Mais on pourrait imaginer plus de méthode si elles nous sont utiles. Ainsi la liste proposée n'est en rien absolue, elle doit être adaptée à vos besoins.

Voici la fonctionnement général d'une des méthodes de la DAO (avec un exemple de code)

```
def create(arme):
    # Etape 1 : On récupère une connexion
    connexion = ReservoirConnexion.getConnexion()
    # Etape 2 : on crée un curseur qui va nous permettre d'exécuter la requête
    curseur = connexion.cursor()
    # Etape 3 : on crée un bloc try/except
    try:
        # Etape 4 : on exécute notre requête SQL. Les %s vont être remplacé par les valeurs passé dans la seconde partie du execute
        curseur.execute(
            "INSERT INTO arme (nom, description_attaque,degat)"
            " VALUES (%s, %s, %s) RETURNING id;"
            , (arme.nom, arme.description_attaque, arme.degat))

        # Etape 5 (optionnelle) : on récupère le résultat de la requête
        arme.id = curseur.fetchone()[0]
        # Etape 6 : on commit notre requête pour la rendre permanente.
        connexion.commit()
        except psycopg2.Error as error:
            # Etape 7 : s'il y a une erreur on fait un rollback et on annule la requête
            AbstractDao.connection.rollback()
            raise error
        finally:
            # Etape 8 : on ferme le curseur pour libérer de la mémoire coté base et on remet la connexion dans notre reservoir à connexion
            curseur.close()
            ReservoirConnexion.putBackConnexion(connexion)
        # Etape 9 : on retourne l'objet demandé.
        return arme
```

Exercice 1 : DAO basique

- Codez les méthodes manquantes de ArmeDao. Respecter les structures suivantes :
    - Méthodes qui mettent à jour des données :

```
# Récupération d'un curseur
cur = self.connection.cursor()
try:
  cur.execute(requete_sql,params)
    # la transaction est enregistrée en base
    self.connection.commit()
    except:
        # la transaction est annulée
        self.connection.rollback()
        raise
    finally:
        cur.close()
return something

```

    - Méthode qui ne font que lire les données :

```
with self.connection.cursor() as cur:
    cur.execute(requete_sql)
    # du code
    return result
```

> 🕮 Rappel des cours sur les base des données. Pour assurer l'intégrité et la cohérence des données, les SGBD ne réalisent pas directement les transactions demandés sur les données. Ils les font d'abord "pour de faux" sans les valider. Et c'est seulement quand une transaction est validée (qu'elle est *commit*) que les changements deviennent permanents. Cela rend les retours en arrière (*rollback*) facile. Et comme nous n'avons pas configurer les auto commit il faut faire le commit à la main. Dans le cas d'opération de lecture, comme on ne modifie pas de donné pas besoin de commit la transaction SQL

- Complétez les tests de la classe TestArmeDao pour qu'ils soit tous au vert. Pourquoi les tests actuels ne sont pas de "bon" tests ? (avec les outils actuellement à votre disposition il est difficile de faire mieux)
- Aller voir dans la base si tout se passe correctement

Exercice 2 : DAO avancée avec des jointures

Pour le moment nos armes ne sont pas complètes, nous ne persistons pas les bonus qu'elles apportent.

- Mettez à jour le fichier `script_bdd.sql` avec la requête de création de la table `statistique_objet` qui va contenir les bonus qu'apportent un objet. Votre table devra avoir une référence vers la table `arme`
- Mettez à jour vos DAO pour manipuler des objets complets (donc avec leurs statistiques)

Pour aller plus loin

- Gestion de la mémoire en python
- Serialisation
- DAO
- psycopg tutorial