M2 CCI - Programmation orientée objet - Java

Examen terminal : Mercredi 6 avril 2022

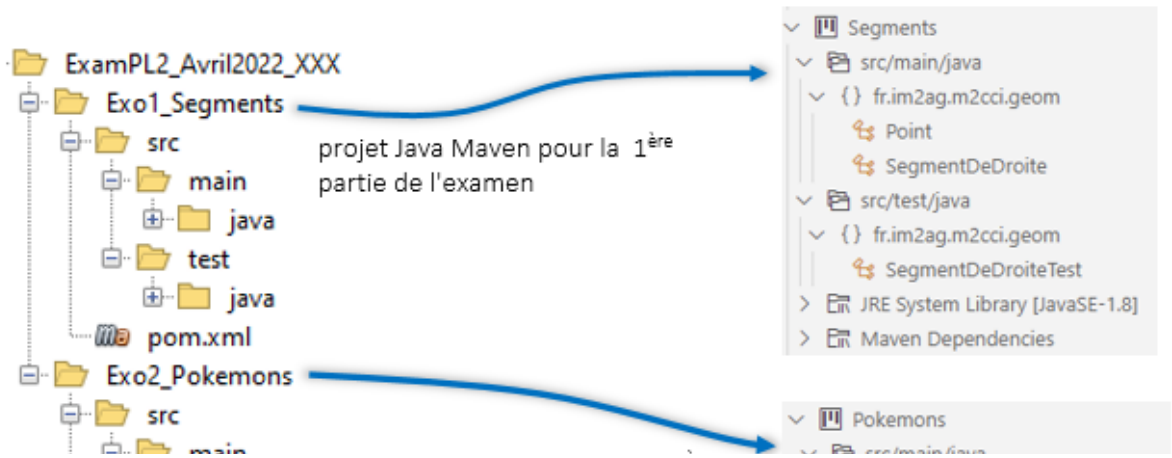
Dernière mise à jour : 05/04/2022 23:06:29 par Philippe.Genoud@imag.fr (mailto:Philippe.Genoud@imag.fr).

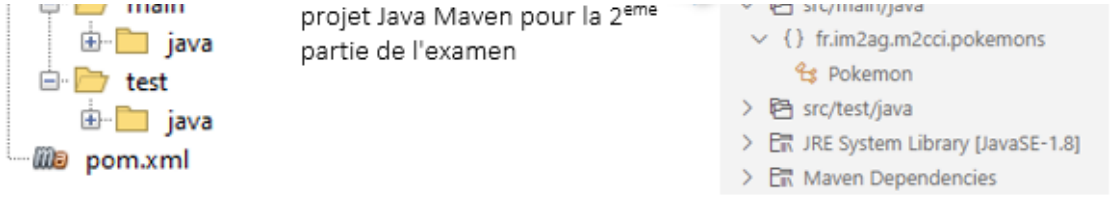
⚠ **Avertissement:** Les documents sont autorisés ainsi que l'accès à internet. Par contre l'utilisation d'outils de messagerie et/ou de discussion en ligne sont **strictement interdits**. De même tout échange de fichiers avec une tierce personne ou tout accès au répertoire d'un autre étudiant sera considéré comme **une fraude** et sanctionné en conséquence. Nous disposons de puissant outils de détections de similarité entre documents afin de prévenir toute tentative de copie.

A la fin de l'épreuve, vous devrez remettre votre travail sous la forme d'un fichier compressé (zip ou tar/gz) que vous déposerez dans l'espace prévu à cet effet sur le serveur moodle de l'ufr IM^{2}AG. gitAfin qu'il n'y ait pas de confusion et de conflits lors de la remise de votre travail vous devrez **OBLIGATOIREMENT** respecter les consignes qui suivent.

Consignes

1. Téléchargez sur votre compte le fichier ExamPL2_Avril2022_XXX.zip (ExamPL2_Avril2022_XXX.zip) et décompressez le. Le répertoire **ExamPL2_Avril2022_XXX** ainsi créé à la structure suivante (fig. 1)





contenu du répertoire `ExamPL2_Avril2022_XXX` .

2. Renommez le répertoire **ExamPL2_Avril2022_XXX** en **ExamPL2_Avril2022_XXX*votreNom***, où ***votreNom*** correspond à votre identifiant de login (sans espaces, ni accents). C'est le contenu de ce répertoire que vous devrez rendre en fin d'épreuve.
3. A la fin de l'épreuve vous déposerez votre travail sur le (serveur moodle (https://im2ag-moodle.e.ujf-grenoble.fr/mod/assign/view.php?id=13279) de l'ufr im2ag)
    1. créez une archive (fichier .tar.gz ou .zip) compressée de votre répertoire de travail.

        - **Soyez très vigilants à ce que tout le travail que vous effectuez durant l'épreuve soit bien enregistré dans ce répertoire**.
        - Pour que les fichiers à transférer ne soient pas trop lourds, veillez à faire un **Clean** sur les différents projets réalisés avant de compresser votre dossier de travail.

    2. Déposez l'archive sur le serveur moodle.
        1. connectez vous sur moodle
        2. si ce n'est déjà fait, enregistrez vous au cours Programmation Internet,
        3. déposez le fichier **ExamPL2_Avril2022_*votreNom*.tar.gz** ou **ExamPL2_Avril2022*votreNom*.zip** précédemment créé dans l'espace prévu à cet effet dans le moodle du cours Applications Internet.

    Pour des explications plus détaillées sur le dépot sur moodle, suivez ce lien (../../AI/depotMoodle/depotFichier.html)

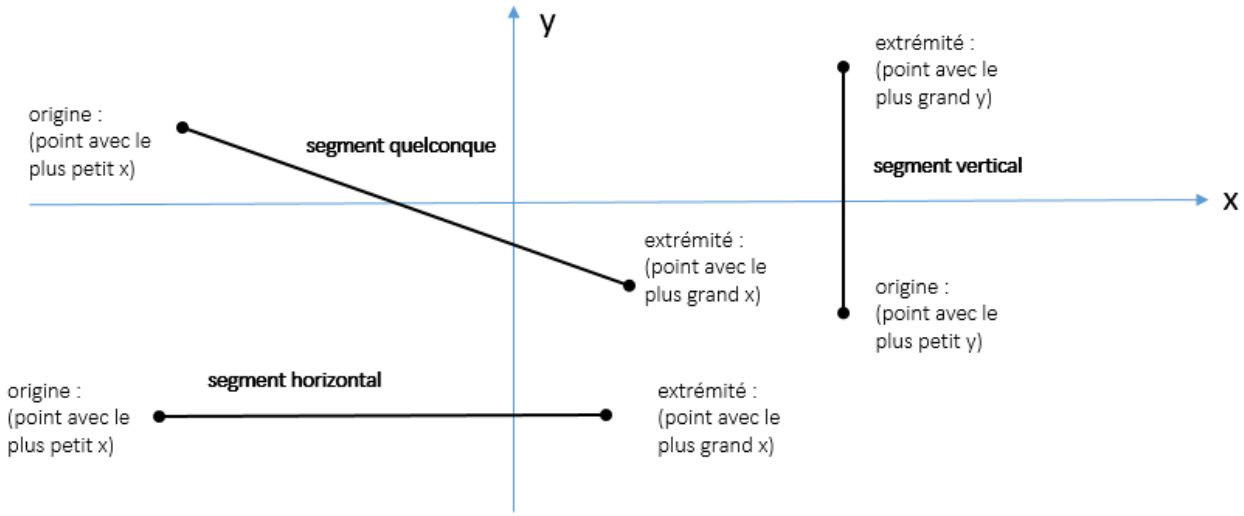
IMPORTANT : Les deux parties sont indépendantes

La première partie du sujet porte sur l'écriture et le test unitaire d'une classe simple. La deuxième partie porte sur l'utilisation des mécanismes objets pour modéliser différents types d'objets (des pokémons).

Partie 1 : Ecriture et test d'une classe simple: **SegmentDeDroite**

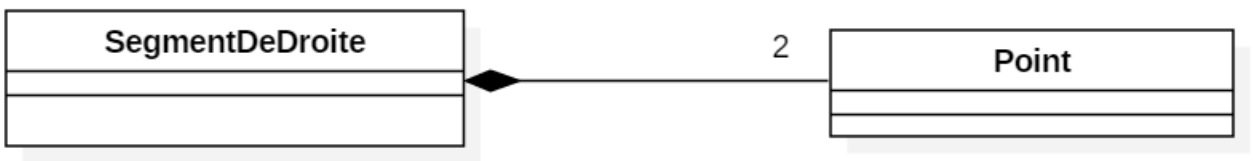
Dans le fichier **Point.java** situé dans le package **fr.im2ag.m2cci.geom** du projet d'application java **Segments** vous disposez d'une classe **Point** (sa **javadoc (../../../../examens/Annales /PL2/M2CCI_PL2_CC_Janvier2020/javadoc/im2ag/m2pcci/geom/Point.html)**). En vous servant de cette classe, écrivez le code java d'une classe **SegmentDeDroite** qui représente un segment de droite dans le plan Oxy et qui respecte les spécifications ci-dessous.

- Un segment de droite est défini par deux points (ses extrémités) qui ne peuvent être confondus
- On appelle $origine$ du segment, celui de ces deux points qui a l'abscisse x la plus petite et si le segment est vertical (abscisses identique) celui qui a l'ordonnée y la plus petite.
- On appelle $extremité$ du segment, celui de ces deux points qui a l'abscisse x la plus grande et si le segment est vertical (abscisses identiques) celui qui a l'ordonnée y la plus grande.



origine et extrémité d'un segment.

- La relation qui relie segment de droite à ses extrémités est une relation de composition forte



relation de composition entre un **SegmentDeDroite** et les **Points** qui le définissent.

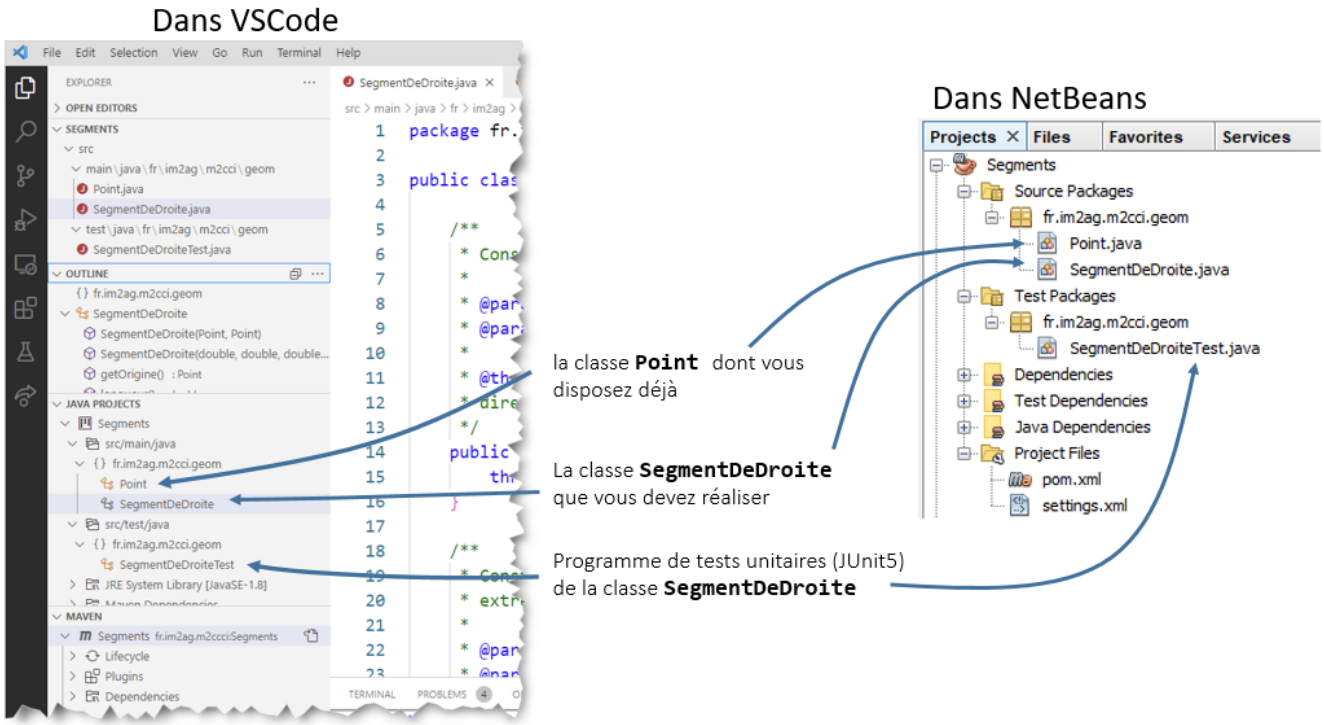
Cela signifie que les 2 points qui définissent le segment ne pourront être modifiés que en passant par les méthodes du segment.

- Les constructeurs proposés par la classe **SegmentDeDroite** permettent de définir un segment en spécifiant ses extrémités
  - soit par leurs coordonnées $(x1, y1)$ et $(x2, y2)$,
  - soit par deux Points $p1$ et $p2$.
  - l'ordre dans lequels ces points sont passés au constructeur n'a pas d'importance. Les instructions $newSegmentDeDroite(x1, y1, x2, y2)$, $newSegmentDeDroite(x2, y2, x1, y1)$, $newSegmentDeDroite(p1, p2)$ ou $newSegmentDeDroite(p2, p1)$ devront toutes produire de objets $SegmentDeDroite$ identiques.
  - ces constructeurs doivent garantir le fait que le segment créé n'est pas "dégénéré", c'est à dire que ses points extrémités ne sont pas confondus. Dans le cas contraire l'objet n'est pas créé et une exception de type **java.lang.IllegalArgumentException** (https://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html) (sous classe de **java.lang.RunTimeException** (https://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html)) est lancée.
- Les méthodes de cette classe sont :
  - **getOrigine** qui retourne un point correspondant à $origine$ du segment.
  - **getExtremite** qui retourne un point correspondant à $extremité$ du segment.
  - **longueur** qui calcule et retourne la longueur du segment.
  - **translater** qui permet de translater le segment dans le plan.
  - **estVertical** et **estHorizontal** qui permettent de tester si le segment est respectivment vertical ou horizontal.
  - **toString** qui renvoie une représentation textuelle (chaîne de caractères) du segment, plus précisément les coordonnées des deux points le définissant, les coordonnées de son $origine$ en premier, celles de son $extrémité$ en second. Par exemple pour un segment défini par les points $P1(12, 24)$ et $P2(32, -7)$, la chaîne retournée par cette méthode sera

    ```
    SegmentDeDroite[(12.0,24.0);(32.0,-7.0)]
    ```

  - **equals** qui permet de tester l'égalité du segment avec un autre segment.

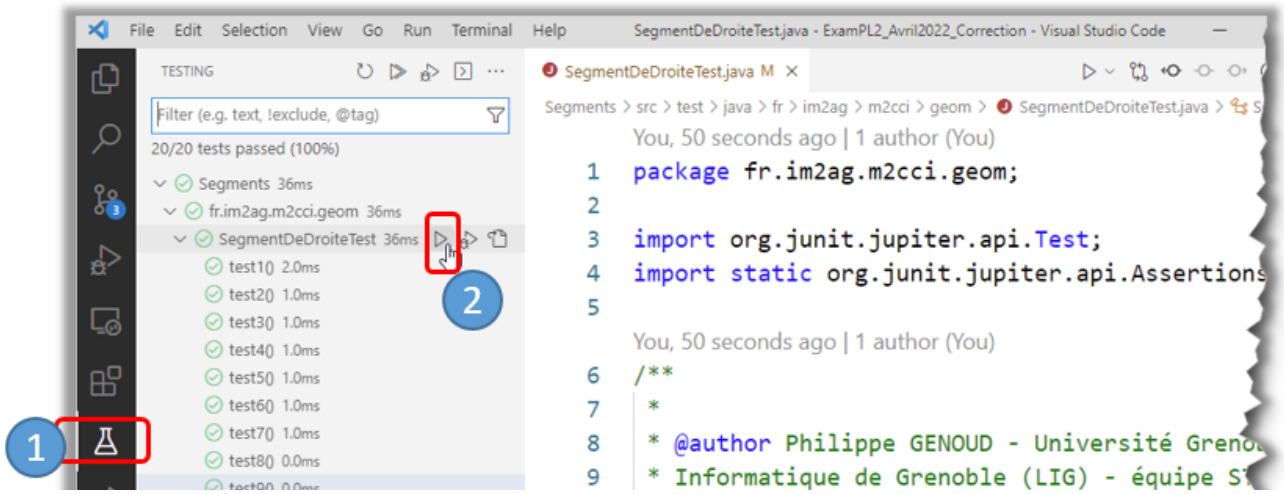
Adepte du TDD (*Test Driven Development* ou en français Développement Piloté par les Tests), le chef de projet qui vous a confié la tâche de coder la classe **SegmentDeDroite** a pris le soin de faire écrire auparavant, par un autre développeur chargé des tests, un programme de tests unitaires (en utilisant le framework Junit 5). Ce programme, **SegmentDeDroiteTest** se trouve dans le dossier **Test Packages** du projet maven **Segments** (voir figure ci-dessous).

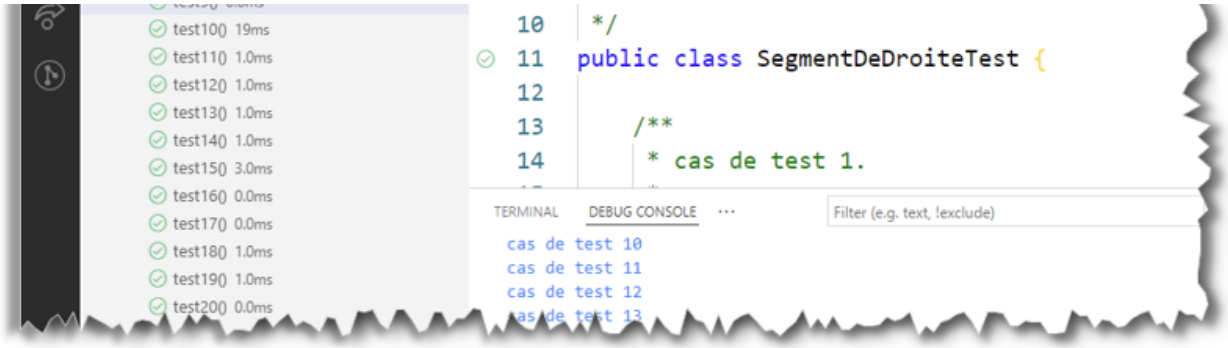


organisation du projet Java maven **Segments**.

Question 1:

En respectant les spécifications ci dessus, complétez le code Java de la classe **SegmentDeDroite** et vérifiez que les tests unitaires définis dans **SegmentDeDroiteTest** sont tous exécutés avec succès. **ATTENTION :** Vous ne devez pas modifier le code de **SegmentDeDroiteTest**.





exécution des tests unitaires du projet **Segments** avec VSCode

cliquer sur le bouton pour voir comment procéder avec NetBeans.

Question 2 :

1. Rajoutez à la classe **SegmentDeDroite** deux méthodes **changerOrigine** et **changerExtremite** qui permettent de remplacer respectivement l'origine ou l'extrémité du segment par un nouveau point.
2. Ecrivez les tests unitaires permettant de vérifier que vos méthodes ont bien l'effet attendu.

Question 3 :

Dans une classe **AppliSegments** ecrivez un programme qui :

1. lit au clavier les coordonnées de deux points définissant un segment,
2. Si le segment est valide (extrémités non confondues) le programme affiche :
   - le point origine du segment,
   - le point extrémité du segment,
   - la longueur du segment,

   sinon programme affiche un message d'erreur et invite l'utilisateur à recommencer.
3. demande à l'utilisateur si il veut recommencer en 1. avec un autre point ou si il veut terminer l'exécution du progamme.

Ci dessous une trace de l'exécution attendue, en jaune les données rentrées par l'utilisateur)

```
donnez les coordonnées (x,y)  du premier point du segment
14 23
donnez les coordonnées (x,y)  du deuxième point du segment
12 45
origine   : (12.0,45.0)
extrémité : (14.0,23.0)
longueur  : 22.090722034374522
Voulez vous continuer O/N ?
O
donnez les coordonnées (x,y)  du premier point du segment
10 10
donnez les coordonnées (x,y)  du deuxième point du segment
10 10
points confondus! recommencez !
donnez les coordonnées (x,y)  du premier point du segment
23 -34
donnez les coordonnées (x,y)  du deuxième point du segment
22 28
origine   : (22.0,28.0)
extrémité : (23.0,-34.0)
longueur  : 62.00806399170998
Voulez vous continuer O/N ?
N
Au revoir !
```

Partie 2 : Une hiérachie de Pokémons

Les Pokémons sont de gentils animaux qui sont passionnés par la programmation objet en général et par le polymorphisme en particulier. Il existe quatre grandes catégories de pokémons :

- Les **pokémons sportifs** : Ces pokémons sont caractérisés par un nom, un $poids$ (en kg), un $nombre\ de\ pattes$, une $taille$ (en mètres) et une $fréquencecardiaque$ mesurée en nombre de pulsations à la minute. Ces pokémons se déplacent sur la terre à une certaine $vitesse$ que l'on peut calculer grâce à la formule suivante :
  $vitesse = nombre\ de\ pattes * taille * 3$
- Les **pokémons casaniers** : Ces pokémons sont caractérisés par un nom, un $poids$ (en kg), un $nombre\ de\ pattes$, une $taille$ (en mètres) et le $nombre\ d'heures\ par\ jour$ où ils regardent la télévision. Ces pokémons se déplacent également sur la terre à une certaine $vitesse$ que l'on peut calculer grâce à la formule suivante :
  $vitesse = nombre\ de\ pattes * taille * 3$

- Les **pokémons des mers** : Ces pokémons sont caractérisés par un nom, un $poids$ (en kg) et un $nombre\ de\ nageoires$. Ces pokémons ne se déplacent que dans la mer à une $vitesse$ que l'on peut calculer grâce à la formule suivante :
  $vitesse = poids/25 * nombre\ de\ nageoires$
- Les **pokémons de croisière** : Ces pokémons sont caractérisés par un nom, un $poids$ (en kg) et un $nombre\ de\ nageoires$. Ces pokémons ne se déplacent que dans la mer à une $vitesse$ que l'on peut calculer grâce à la formule suivante :
  $vitesse = (poids/25 * nombre\ de\ nageoires)/2$

Question 1 : Sur votre copie dessiner un diagramme de classes permettant de gérer la hierarchie des pokémons.

Pour chacune de ces quatre catégories de pokémons, on désire disposer d'une méthode $toString$ qui retourne (dans une chaîne de caractères) les caractéristiques du pokémon. Par exemple

- appliquée sur un **pokémon sportif** cette méthode retournera
  "Je suis Pikachu mon poids est de 18 kg, ma vitesse est de 5.1 km/h, j'ai 2 pattes, ma taille est de 0,85m, ma fréquence cardiaque est de 120 pulsations à la minute"
- appliquée sur un **pokémon casanier** cette même méthode retournera
  "Je suis Salameche mon poids est de 12 kg) ma vitesse est de 3.9 km/h, j'ai 2 pattes, ma taille est de O.65 m, je regarde la télé 8h par jour"
- sur un **pokémon des mers** :
  "Je suis Rondoudou mon poids est de 45 kg, ma vitesse est de 3.6 km/h, j'ai 2 nageoires"
- et enfin sur un **pokémon de croisière**
  "Je suis Bulbizarre mon poids est de 15 kg, ma vitesse est de 0,9 km/h, j'ai 3 nageoires"

Question 2 : Programmez en Java la ou les classes représentant les différents types de pokémons.

On désire maintenant définir une classe $CollectionPokemons$ qui comporte en attribut privé une liste pouvant contenir des pokémons de catégories quelconques. En plus d'un constructeur, cette classe doit comprendre les méthodes suivantes :

- une méthode qui insère un pokémon dans la collection
- une méthode qui calcule la vitesse moyenne des pokemons de la collection.

De plus on souhaite que la collection de pokémons soit itérable, cette à dire qu'elle possède une méthode qui retourne un itérateur permettant de parcourir la liste des pokémons. Par exemple si $maCollection$ est une collection de pokémons, pour afficher l'ensembles des pokémons qu'elle

contient on veut pouvoir écrire le code Java suivant

```
for (Pokemon p : maCollection) {
   System.out.println(p);
}
```

Question 3 : Ecrire la classe $CollectionPokemons$.

Question 4 : Ajoutez à la classe $CollectionPokemons$ une méthode $vitesseMoyenneSportifs()$ qui retourne la vitesse moyenne des pokémons sportifs de la collection.

Question 5 : Ecrire un programme de test la classe $CollectionPokemons$.

via GIPHY (https://giphy.com/gifs/pokemon-friendship-high-five-10LKovKon8DENq)