TP N°1 *Diagrammes de classes et coder en Java*

Objectifs

- UML : Diagrammes de classe simples
- BASH : savoir compiler et exécuter du code Java
- JAVA : Comprendre et écrire du code JAVA simple

Exercice 1 *Echauffement*

1.1 Dans un dossier `tp1/pokemon/` crée le fichier `Pokemon.java` dans lequel tu copieras le code suivant. Quelle commande bash permet de compiler ce fichier ? Vérifie qu'il compile sans erreur.

```
public class Pokemon {
    private String nom;
    private int force;
    public Pokemon(String nomPokemon, int forcePokemon) {
        this.nom = nomPokemon;
        this.force = forcePokemon;
    }
    public Pokemon(String nomPokemon) {
        this(nomPokemon, 10);
    }
    public String getNom() {
        return this.nom;
    }
    public int getForce() {
        return this.force;
    }
    public void evolue(String newName, int newForce) {
        this.nom = newName;
        this.force = newForce;
    }
    public void evolue(String newName) {
        this.evolue(newName, this.force + 10);
    }
    @Override
    public String toString() {
        return this.nom +"(force "+ this.force + ")";
    }
}
```

1.2 Dans le dossier `tp1/pokemon/` crée le fichier `ExecutablePokemon.java` dans lequel tu copieras le code suivant. Vérifie qu'il compile sans erreur. Quelle commande bash permet d'exécuter cette classe ? Vérifie qu'elle s'exécute sans erreur.

```
public class ExecutablePokemon {
   public static void main(String [] args) {
       Pokemon poke;
       poke = new Pokemon("Bulbizarre", 30);
       poke.evolue("Herbizarre", 37);
       poke.evolue("Florizarre");
       System.out.println(poke.toString());  // (1)
   }
}
```

1.3 Explique l'affichage obtenu à la ligne signalée (1)

1.4 Dans cet exécutable, ajoute la création d'un pokemon Abo dont la force est égale à 10, puis fais-le évoluer en Arbok dont la force est de 24. Fais afficher les informations sur ce pokemon.

Exercice 2 *Une classe pour modéliser un couple de deux nombres entiers*

2.1 Voici le code java de deux classes :

```java
public class Couple {

    private int premier;
    private int second;

    public Couple(int x, int y) {
        this.premier = x;
        this.second = y;
    }

    public Couple() {
        this(0, 0);
    }

    public void setPremier(int premier) {
        this.premier = premier;
    }

    public void permute() {
        int aux;
        aux = this.premier;
        this.premier = this.second;
        this.second = aux;
    }

    public int somme() {
        return this.premier + this.second;
    }

    @Override
    public String toString() {
        return "(" + this.premier +", "+ this.second + ")";
    }
}
```

```java
public class ExecutableCouple {
   public static void main(String [] args) {
       Couple unCouple = new Couple(5, -4);
       System.out.println(unCouple.toString());  // (1)
       System.out.println(unCouple.somme());     // (2)
       Couple unAutreCouple = new Couple();
       unAutreCouple.setPremier(7);
       unAutreCouple.permute();
       System.out.println(unAutreCouple.toString());  // (3)
   }
}
```

Lecture de code sans ordinateur

2.2 Sur ces deux classes, seule la classe `ExecutableCouple` est exécutable. Pourquoi ?

2.3 Identifie les attributs, constructeurs et méthodes de la classe `Couple`.

2.4 Précise l'affichage provoqué par les lignes (1), (2) et (3).

2.5 Quelle commande bash permet de compiler ce projet ?

2.6 Quelle commande bash permet d'exécuter ce projet ?

Avec un ordinateur

2.7 Dans un dossier `tp1/couple/` crée un fichier pour chacune des classes proposées (fichiers dans lesquels tu copieras le code donné). Attention, je te rappelle que tu dois nommer tes fichiers correctement sans quoi tu auras des erreurs à la compilation ! Vérifie que le code compile et s'exécute sans erreur. Vérifie ta réponse à la question **2.4**.

2.8 Dans la classe `Couple`, ajoute une méthode `produit()` qui renvoie le produit des deux composantes du couple.

2.9 Dans la classe `ExecutableCouple`, ajoute du code qui permet de visualiser le résultat de la méthode que tu viens de coder puis vérifie le résultat en exécutant ton code (n'oublie pas de le compiler auparavant !)

2.10 Dans la classe `Couple`, ajoute un constructeur qui prend un seul paramètre et qui permet de construire un couple dont les deux composantes sont identiques.

2.11 Dans la classe `ExecutableCouple`, ajoute du code qui utilise le constructeur que tu viens de coder puis vérifie le résultats en exécutant ton code.

2.12 Dans la classe `ExecutableCouple`, ajoute le code suivant qui permet de faire quelques tests. Compile-le et exécute-le. Obtiens-tu le résultat attendu ?

```
public class ExecutableCouple {
   public static void main(String [] args) {
       // Tests pour les méthodes somme() et produit() de Couple
       Couple exemple;
       exemple = new Couple(3, -8);
       assert exemple.somme() == -15;  // FAUX !!!
       assert exemple.produit() == -24;
       exemple = new Couple();
       assert exemple.somme() == 0;
       assert exemple.produit() == 0;
       exemple = new Couple(7);
       assert exemple.somme() == 14;
       assert exemple.produit() == 49;
   }
}
```

2.13 Exécute ton code en ajoutant l'option `-ea` [1] et vérifie que tu as bien le résultat attendu.

2.14 Corrige le test incorrect et vérifie que les tests passent tous correctement (n'oublie pas l'option `-ea` !!!)

Nous verrons plus tard comment écrire "correctement" des tests mais pour le moment nous nous contenterons de vérifier nos méthodes de cette façon dans un executable.

1. `-ea` est la contraction de `-enableassertions`

Exercice 3 *Une classe pour modéliser un personnage de la Terre du Milieu*
On donne le diagramme de classe d'une classe `Personnage` :

Personnage
- nom:String - tailleBarbe:int - tailleOreilles:int
+ Personnage(nom:String, barbe:int, oreille:int) + getNom():String + getBarbe():int + getTailleOreilles():int

3.1 Dans un dossier `tp1/personnage`, crée le fichier `Personnage.java` dans lequel tu coderas la classe `Personnage`

3.2 Vérifie que ton fichier compile sans erreur.

3.3 Dans le dossier `tp1/personnage` ajoute le fichier `ExecutablePersonnage.java` dont on donne le contenu. Vérifie que ce code compile et s'exécute sans erreur.

```
public class ExecutablePersonnage{
    public static void main(String [] args){
        Personnage nain = new Personnage("Gimli", 65, 15);
        System.out.println(nain.getNom());
        System.out.println(nain.getTailleOreilles());
        System.out.println(nain.getBarbe());
        // Tests
        // A compléter
  }
}
```

3.4 Complète l'exécutable de manière à tester les méthodes `getTailleOreilles()` et `getBarbe()`. Vérifie que ton code passent les tests.

⚠ Les assertions en java

Par défaut, les assertions ne sont pas activées en java. Donc si vous lancez l'application, sans activer les assertions, aucune erreur ne sera produite en cas de violation d'assertion.
Pour activer les assertions, on ajoute l'option courte `-ea` ou l'option longue `-enableassertions` à la commande `java`