CHAPUS 15/05/2023

TP n°2 Java

Louka 3A IE

Exercice 1 :

2./

Classe Personne

```java
public class Personne {
    private String nom ;
    public String prenom ;
    protected int age ;

    public Personne (){
        nom = "";
        prenom = "";
        age = -1;
    }
    public Personne ( String nom , String prenom , int age ){
        this . nom = nom ;
        this . prenom = prenom ;
        this . age = age ;
    }
}
```

Classe Personnel

```java
public class Personnel extends Personne {
    private String service;


    public Personnel (String service, String nom , String prenom , int age ){
    super(nom, prenom, age);
    this . service = service ;
    }

    public void afficherAge () {
        System .out . println ( "Ce personnel a " + age + " an(s)" );
    }
}
```

3./

La fonction super() permet d'appeler le constructeur de la classe mère.

4./

Classe Main

```
class Main
{
    public static void main(String[] args) {
        Personnel p1 = new Personnel("Informatique","Chapus","Louka",22);
        p1.afficherAge();
    }
}
```

```
Ce personnel a 22 an(s)
```

Après exécution de la class Main on a bien l'âge du personnel qui s'affiche.

5./

Lorsqu'on essaye d'affiche le nom on a une erreur car la variable nom est en accès privé.

```
Personnel.java:14: error: nom has private access in Personne
                System .out . println ( "Ce personnel s'appele " + nom );
                                                                  ^
1 error
```

En revanche pour le prénom la fonction fonctionne et on arrive effectivement à afficher le prénom du personnel.

```
Ce personnel a 22 an(s)
Ce personnel a pour prénom Louka
```

Donc dans une classe fille on peut accéder aux variables protected et public mais pas celle définie en private.

6./

Oui une classe fille peut être une classe de base car par la suite on devra définir deux autres classes (Administratif et Enseignant) qui sont toutes les deux des classes fille de la classe Personnel qui est elle-même classe fille de la classe Personne.

7./

On défini toutes les classes avec les getters et setters en respectant la hiérarchie donné et en mettant toutes les variables de classe en protected pour toujours y avoir accès.

Exercice 2 :

1./

```
class Execution
{
    public static void main(String[] args) {

    }
}
```

Classe Exécution

La classe Exécution contient la fonction main() qui est vide pour l'instant.

2./

```
class Execution
{
    public static void main(String[] args) {
        Etudiant e1 = new Etudiant("Pascal",4,"Dupont","Alice",21);
        Etudiant e2 = new Etudiant("Ohm",0,"Dujardin","Benjamin",22);
        Etudiant e3 = new Etudiant("Promotion inconnue",0,"Anonyme","Anonyme",-1);
        Etudiant e4 = new Etudiant("Promotion inconnue",0,"Anonyme","Anonyme",-1);

        Administratif a1 = new Administratif("secrétaire","informatique","Dupont","Béatrice",19);

        Enseignant ens1 = new Enseignant("informatique","électronique", "info", "Machin","Boris",36);
    }
}
```

On déclare tous les objets comme demandé.

3./

```
public void ouMeTrouver(){
    System.out.println("Je ne sais pas où je suis.");
}
```

Fonction ouMeTrouver

La fonction est simplement définie avec un println.

Exercice 3 :

1./

```
e1.ouMeTrouver();
a1.ouMeTrouver();
ens1.ouMeTrouver();
```

Lorsqu'on essaye d'appeler la fonction ouMeTrouver de chaque objet il y a un message d'erreur car la fonction n'est pas définie dans chaque classe.

2./

```
Je suis en cours tous les jours, sauf les jeudis après-midis (je fais du sport, mais c'est comme un cours car j'aime le sport).
Le plus facile c'est de passer à mon bureau.
Ce n'est pas la peine de me chercher, je saurai vous trouver !
```

Résultat après exécution

Le code de la fonction exécuter n'a pas changé, on a juste ajouté des fonctions ouMeTrouver dans chaque classe.

4./

```
Personne tab[] = new Personne[5];
tab[0] = e1;
tab[1] = e2;
tab[2] = e3;
tab[3] = a1;
tab[4] = ens1;
```

On créer un tableau de personnes car c'est la classe que tous les objets ont en commun.

5./

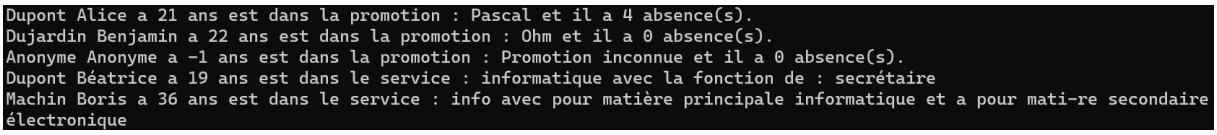
Comme on a un tableau de Personne, alors si on appeler la fonction ouMeTrouver sans vérifier le type on aurait toujours le même résultat à savoir celui de la classe Personne.

6./

```
for(int k=0;k<5;k++){
    System.out.println(tab[k].toString());
}
```

Affichage de tous les éléments du tableau

On redéfini la méthode toString dans chaque classe pour être sur que c'est la bonne méthode qi est appeler.



Résultat après exécution

On voit bien qu'à chaque fois le texte est différent, donc on appelle bien la bonne méthode toString de la bonne classe. On n'a même pas besoin de vérifier le type de chaque objet.