Atelier 2: Programmation avancée (Java)

IssatMa

Sections: Tic et EEA : 1ère année.

Objectif du TP:

Ce TP vise à explorer les concepts de programmation orientée objet en Java à travers la réalisation d'une mini application assistée étape par étape.

Notions à voir:

- Classe / Objet
- Attributs
- Méthodes
- Constructeurs
- surcharge de constructeurs
- surcharge de méthodes

Étape 1:

Dans votre IDE (intellij en TP), ouvrir un nouveau projet java.

1. sous **src**, créer une nouvelle classe appelée Employe
2. dans le fichier Employe.java nouvellement crée, saisir le code suivant:

```
public class Employe {
    // attributs
    public String matricule;
    public double salaireDeBase;
    public double coutHeure;
    //méthodes
    public double calculerSalaire(int heuresExtra){
        return salaireDeBase+coutHeure*heuresExtra;
    }
}
```

3. revenir à la classe Main, puis créer un objet Employe en lui affectant un salaire de base, un coût heure supplémentaire (coutHeure) et afficher son salaire.

```
public class Main {

   public static void main(String[] args) {

        Employe aicha = new Employe();

        aicha.matricule = "25E89";

        aicha.salaireDeBase = 1020;

        aicha.coutHeure = 18;

        double salaireComplet = aicha.calculerSalaire(10);

        System.out.println("matricule employé: " +
aicha.matricule);

        System.out.println("salaire = "+ salaireComplet);


   }
}
```

Étape 2: Encapsulation et abstraction

→ encapsulation: mettre le code dans des capsules séparées.
→ abstraction: ne laisser public que ce qui est nécessaire à une autre Classe de voir.

modifier le code de la classe Employe:

```
public class Employe {
   private String matricule;
   private int salaireDeBase;
   private int coutHeure;

   public String getMatricule() {
      return matricule;
   }

   public void setMatricule(String matricule) {
      this.matricule = matricule;
   }

   public int calculerSalaire(int heuresExtra) {
       return salaireDeBase + getCoutHeure() * heuresExtra;
   }

   public void setSalaireDeBase(int salaireDeBase) {
       if (salaireDeBase <= 0) {
           throw new IllegalArgumentException("Le Salaire ne peut pas
etre négatif !!!");
```

```
        } else this.salaireDeBase = salaireDeBase;
    }

    public int getSalaireDeBase() {
        return salaireDeBase;
    }

    private int getCoutHeure() { // abstraction.
        return coutHeure;
    }

    public void setCoutHeure(int coutHeure) {
        if (coutHeure > 0) {
            this.coutHeure = coutHeure;
        } else {
            throw new IllegalArgumentException("coutHeure ne peut pas
etre négatif !!!");
        }
    }
}
```

Dans la classe **Main,** on aura plus accès direct aux attributs, on passera donc seulement par les méthodes publiques:

```
public class Main {

    public static void main(String[] args) {

        Employe aicha = new Employe();

        aicha.setMatricule("25E89");

        aicha.setSalaireDeBase(1020);

        aicha.setCoutHeure(18);

        double salaireComplet = aicha.calculerSalaire(10);

        System.out.println("matricule employé: " +
aicha.getMatricule());

        System.out.println("salaire = "+ salaireComplet);


    }

}
```

Étape 3: Les Constructeurs:

Dans cet exemple, on **CONSTRUIT** un nouvel employé par:

```
Employe aicha = new Employe();
```

→ Constructeur par défaut

* Si on veux passer les paramètres nécessaires à la construction d'un nouvel employé directement via le constructeur:

→ Constructeur personnalisé:

```
public Employe(String matricule, int salaireDeBase, int coutHeure) {
   this.matricule = matricule;
   this.salaireDeBase = salaireDeBase;
   this.coutHeure = coutHeure;
}
```

N.B: la méthode setSalaireDeBase sera transformée en mode private, plus besoin d'y accéder en dehors de la classe Employe.

* Dans la classe Main, le code sera modifié en conséquence:

```
public class Main {
   public static void main(String[] args) {
      Employe aicha = new Employe("25E89", 1020, 18);
      System.out.println("matricule employé: " + aicha.getMatricule());
      System.out.println("salaire = "+  aicha.calculerSalaire(10));

   }
}
```

Étape 4: Surcharge de méthodes / constructeurs

- Cas d'étude: Employe sans heures Extra:
  La méthode calculerSalaire de la classe Employe prend un argument heuresExtra:
  `public int calculerSalaire(int heuresExtra)`

  Si un employé n'a pas de salaire?
  → solution 1: mettre 0 comme argument.
  → solution 2: créer une autre méthode calculerSalaire.
- Dans la classe Employe, ajouter le code suivant:

```
public int calculerSalaire(){
   return salaireDeBase;
}
```

- On devrait alors créer un autre Constructeur (autre façon de créer un Employe) sans heuresExtra:
-

```
public Employe(String matricule, int salaireDeBase) {
   this.matricule = matricule;
   this.salaireDeBase = salaireDeBase;
}
```

Étape 5: Les attributs et méthodes statiques

un attribut ou méthode static ne dépend pas d'un objet mais de la classe elle-même et il est partagé à tous les objets de cette classe.

→ c'est un attribut de classe

- Cas d'étude: nombre d'employés:

pour calculer nombreDemployees:

1. ajouter après les attributs: `public static int nombreDemployees;`
2. ajouter nombreDemployees++; aux deux constructeurs de la classe Employe
3. créer une méthode static pour retourner cet attribut static:

```
public static int getNombreDemployees(){
    return nombreDemployees;
}
```

4. Appeler cette méthode statique pour afficher le nombre d'employés dans la classe Main.