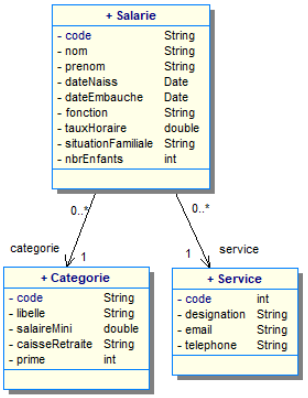
2023/2024	Projet Java – Etape 02
BTS SIO 1SIOB - SLAM	Auteur : AUGEREAU Eliott et PORTOLLEAU Anaïs Date de rédaction : 16/04/2024

Étape n°2 – compléter les classes métier

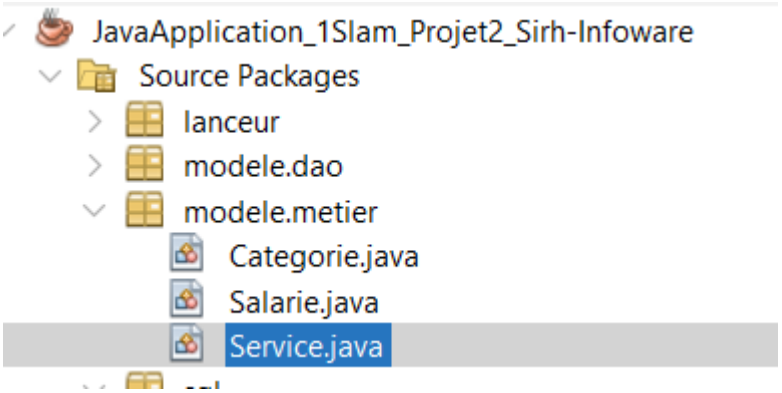


Voici un extrait du diagramme des classes métier de l'application établi par votre tuteur :

Les classes Salarie et Service ont déjà été codées.

Travail à faire :
Testez les classes métier existantes à l'aide des classes de tests unitaires correspondantes (paquetage test.metier).

On a bien testé les deux classes de test dans le package metier comme on peut voir ci-dessous :



Voici l'exécution de TestSalarie :

```
Output - JavaApplication_1Slam_Projet2_Sirh-Infoware (run) #2
run:
TestSalarie
1 - Salari sans service
Salarie{code=R06, nom=LANDREAU, prenom=Bertrand, dateNaiss=12/12/1980  , dateEmbauche=15/11/2006  , fonction=D veloppeur, tauxHorai
 service=N ant}
2 - Salari avec service
Salarie{code=R06, nom=LANDREAU, prenom=Bertrand, dateNaiss=12/12/1980  , dateEmbauche=15/11/2006  , fonction=D veloppeur, tauxHorai
 service=Service{code=1, designation=Informatique, email=Inf-logihome@logihome.com, telephone=0169983212}}
BUILD SUCCESSFUL (total time: 0 seconds)
```

Voici l'exécution de TestService :

```
Output - JavaApplication_1Slam_Projet2_Sirh-Infoware (run) #2
run:
TestService
Service{code=1, designation=Informatique, email=Inf-logihome@logihome.com, telephone=0169983212}
BUILD SUCCESSFUL (total time: 0 seconds)
```

On peut voir que les deux classes de Test s'exécutent correctement.

Vous devez coder en java la **classe métier Catégorie**, ainsi que sa **classe de test unitaire,** et montrer que les tests sont conformes aux attentes.

Voici donc le code requis:

Nous avons donc ici créer la classe “Categorie” avec les attributs demandés,

```
 * Classe métier Service
 * @author btssio
 */
public class Categorie {
    private int code;
    private String libelle;
    private double salaireMini;
    private String CaisseRetraite;
    private int prime;
    /**
     * Constructeur simple
     * @param code
     * @param libelle
     */
```

3 constructeurs différents,

ainsi qu'avec les méthodes “Get”, “Set” et “toString” pour chaque attribut:

```
public double getSalaireMini() {
    return salaireMini;
}

public void setSalaireMini(double salaireMini) {
    this.salaireMini = salaireMini;
}

public String getCaisseRetraite() {
    return CaisseRetraite;
}

public void setCaisseRetraite(String CaisseRetraite) {
    this.CaisseRetraite = CaisseRetraite;
}

public int getPrime() {
    return prime;
}

public void setPrime(int prime) {
    this.prime = prime;
}

@Override
public String toString() {
    return "Categorie{" + "code=" + code + ", libelle=" + libelle + ", salaireMini=" +
}
```

Voici sa classe de TestUnitaire :
Nous y testons donc les différentes méthodes définies dans le code “Categories.java”

```java
package test.metier;
import modele.metier.Categorie;


/**
 * Classe de test unitaire de la classe Categorie
 * @version 2024
 */
public class TestCategorie {


    public static void main(String[] args) {
        System.out.println("TestCategorie");


        // Test des constructeurs
        testConstructeurs();


        // Test des accesseurs et mutateurs
        testAccesseursMutateurs();
    }


    public static void testConstructeurs() {
        System.out.println("Test des constructeurs :");


        // Création de différentes instances de Categorie avec les constructeurs
        Categorie categorie1 = new Categorie();
        Categorie categorie2 = new Categorie(1, "Catégorie A");
        Categorie categorie3 = new Categorie(2, "Catégorie B", 2000.0, "Caisse B", 100);
```

```
        // Affichage des informations des catégories
        System.out.println(categorie1.toString());
        System.out.println(categorie2.toString());
        System.out.println(categorie3.toString());
    }


    public static void testAccesseursMutateurs() {
        System.out.println("Test des accesseurs et mutateurs :");


        // Création d'une instance de Categorie
        Categorie categorie = new Categorie();


        // Utilisation des mutateurs pour définir les attributs
        categorie.setCode(3);
        categorie.setLibelle("Catégorie C");
        categorie.setSalaireMini(1500.0);
        categorie.setCaisseRetraite("Caisse C");
        categorie.setPrime(50);


        // Utilisation des accesseurs pour récupérer les attributs et les afficher
        System.out.println("Code : " + categorie.getCode());
        System.out.println("Libelle : " + categorie.getLibelle());
        System.out.println("Salaire Minimum : " + categorie.getSalaireMini());
        System.out.println("Caisse Retraite : " + categorie.getCaisseRetraite());
        System.out.println("Prime : " + categorie.getPrime());
    }
```

Complétez la classe Salarie pour tenir compte de l'association

Nous avons d'abord importé dans le fichier "Salaire.java" afin de pouvoir effectuer l'association : (Or, il n'est pas indispensable, car ils sont présents dans le même paquetage.)

```
import modele.metier.Categorie;
```

Puis créé l'association ManyToOne on fait ceci on déclare un objet de type Categorie qui s'appelle categorie:

```
private Categorie categorie;
```

Voici la vue d'ensemble Salarie-Categorie ;

```java
import java.util.Date;
import java.util.Locale;
import modele.metier.Categorie;
import modele.metier.Service;

/**
 * Classe métier
 * @author btssio
 */
public class Salarie {

    private String code = "";
    private String nom = "";
    private String prenom = "";
    private Date dateNaiss;
    private Date dateEmbauche;
    private String fonction;
    private double tauxHoraire;
    private String situationFamiliale;
    private int nbrEnfants;
    private Service service;
    private Categorie categorie;
```

Puis on a modifié le constructeur du salarié en ajoutant la catégorie. On a mit la catégorie à null parce qu' il n'est pas définie et dans le deuxième constructeur vu qu'il est définit on utilise this

1er Constructeur:

```java
ublic Salarie(String code, String nom, String prenom, Date dateNaiss, Date dateEmbauche, String fonction, double tauxHoraire, String situationFamilia
    this.code = code;
    this.nom = nom;
    this.prenom =prenom;
    this.dateNaiss = dateNaiss;
    this.dateEmbauche = dateEmbauche;
    this.fonction = fonction;
    this.tauxHoraire = tauxHoraire;
    this.situationFamiliale = situationFamiliale;
    this.nbrEnfants = nbrEnfants;
    this.service = null;
    this.categorie = null;
```

2ème Constructeur: (Ici nous appelons la classe Categorie, donc nous devons le définir.)

```
public Salarie(String code, String nom, String prenom, Date dateNaiss, Date dateEmbauche, Stri
    this( code,  nom, prenom,  dateNaiss,  dateEmbauche,  fonction,  tauxHoraire,  situationFar
    this.service = serv;
    this.categorie = cat;
```

Nous rajoutons donc également un “Get”, un “Set” et un “toString” pour la classe Catégorie:

```
    }

    public Categorie getCategorie() {
        return categorie;
    }

    public void setCategorie(Categorie categorie) {
        this.categorie = categorie;
    }
```

Nous modifions aussi le “toString”, en important le “toString” de la classe “categorie.java”, et en l’utilisant en le définissant dans le “toString” de “Salarie.java”:

```
@Override
public String toString() {
    String dateNaissFmt = String.format("%1$td/%1$tm/%1$tY  ", dateNaiss);
    String dateEmbFmt = String.format("%1$td/%1$tm/%1$tY  ", dateEmbauche);
    String txHoraireFmt = String.format(Locale.FRANCE, "%1$5.2f", tauxHoraire);
    String serviceToString = (service == null ? "Néant" : service.toStringEtat());
    String categorieToString = (categorie == null ? "Néant" : categorie.toString());
    return "Salarie{" + "code=" + code + ", nom=" + nom + ", prenom=" + prenom + ", dateNaiss=" + dateNaissFmt + ", dateEmbauche=" + dateEmb
            + ", fonction=" + fonction + ", tauxHoraire=" + txHoraireFmt + ", situationFamiliale=" + situationFamiliale + ", nbrEnfants=" +
            + ",\n service=" + serviceToString + ", \n categorie" + categorieToString +'}';

}
```

des erreurs de conception à cette étape pouvant affecter de façon importante la suite de votre développement, il vous est conseillé de montrer votre code à un enseignant avant de commencer l’étape suivante.
On a bien montré notre code et il a été validé.

2023/2024	Projet Java – Etape 02
BTS SIO 1SIOB - SLAM	Auteur : AUGEREAU Eliott et PORTOLLEAU Anaïs Date de rédaction : 16/04/2024

Modifiez la classe de test unitaire de Salarie pour tenir compte de l'évolution de cette classe et exécutez la à nouveau.

Pour tenir compte de l'évolution de la classe, nous avons donc créer un objet de type "catégorie" dans le ficher test, afin de bien simuler et de bien coller avec le constructeur de "salarie", nous créons ensuite différents tests pour chaque situation: (avec service et catégorie, sans service et catégorie, avec l'un et pas l'autre…)

Test 1: salarié sans service et sans catégorie:

```
System.out.println("1 - Salarié sans service et sans catégorie");
try {
    unSalarie = new Salarie("R06", "LANDREAU", "Bertrand", sdf.parse("12/12/1980"), sdf.parse("15/11/2006"), "Développeur", 10.0, "marié", 2);
} catch (ParseException ex) {
    ex.printStackTrace();
}
System.out.println(unSalarie.toString());
```

Test 2: Salarié avec service et catégorie

```
    System.out.println("2 - Salarié avec service et catégorie");
    try {
        Service serv = new Service(1, "Informatique", "Inf-logihome@logihome.com", "0169983212");
        Categorie cat =new Categorie(35,"Ordinateur", 25528465, "ARRCO",10);
        unSalarie = new Salarie("R06", "LANDREAU", "Bertrand", sdf.parse("12/12/1980"), sdf.parse("15/11/2006"), "Développeur", 10.0, "marié", 2, serv, cat);
    } catch (ParseException ex) {
        ex.printStackTrace();
    }
    System.out.println(unSalarie.toString());
```

Test 3: Salarié sans service et avec catégorie

```
System.out.println("3 - Salarié sans service et avec catégorie");
try {
    Categorie cat =new Categorie(35,"Ordinateur", 25528465, "ARRCO",10);
    unSalarie = new Salarie("R06", "LANDREAU", "Bertrand", sdf.parse("12/12/1980"), sdf.parse("15/11/2006"), "Développeur", 10.0, "marié", 2,null,cat);
} catch (ParseException ex) {
    ex.printStackTrace();
}
System.out.println(unSalarie.toString());
```

Test 4: Salarié avec service et sans catégorie

```
System.out.println("4 - Salarié avec service et sans catégorie");
try {
    Service serv = new Service(1, "Informatique", "Inf-logihome@logihome.com", "0169983212");
    unSalarie = new Salarie("R06", "LANDREAU", "Bertrand", sdf.parse("12/12/1980"), sdf.parse("15/11/2006"), "Développeur", 10.0, "marié", 2,serv,null);
} catch (ParseException ex) {
    ex.printStackTrace();
}
System.out.println(unSalarie.toString());
```

Nous obtenons donc un résultat Conclusif:

```
TestSalarie
1 - Salari sans service et sans cat gorie
Salarie{code=R06, nom=LANDREAU, prenom=Bertrand, dateNaiss=12/12/1980  , dateEmbauche=15/11/2006  , fonction=D veloppeur, tauxHoraire=10,00, situationFamiliale=mari, nbrEnfants=2,
 service=N ant,
 categorieN ant}
2 - Salari  avec service et cat gorie
```

Bilan de l'étape :
On a pas eu beaucoup de difficultés juste on a passé un peu plus de temps à comprendre comment était organisé le code et l'association qui était nécessaire.

Contraintes :
Une classe métier doit fournir (a minima) les méthodes suivantes :
o un **constructeur** avec des **paramètres** permettant de valoriser les attributs ;
o un **accesseur et un mutateur** pour **chaque attribut** ;
o une **méthode toString** permettant de **contrôler l'état** d'une **instance de la classe**.

Les classes métier sont regroupées dans un paquetage modele.metier .

Les classes de test unitaire sont regroupées dans un paquetage test.metier .

A remettre à l'issue de l'étape:
Dans une archive zip respectant la nomenclature :
le répertoire de votre projet NetBeans ; le code source est normalisé et commenté ;
un compte-rendu de l'étape comportant les éléments suivants :
o vos explications ;
o un rapport de tests unitaires (trace d'exécution des classes de test commentée) ;
o un bilan de l'étape (fait, non fait, difficultés rencontrées).