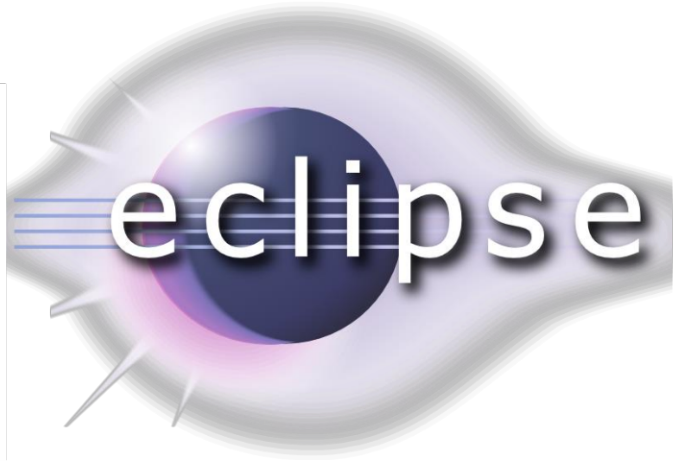
Université Hassan 1er
École Nationale des Sciences Appliquées – Berrechid

POLYCOPIE DES TRAVAUX PRATIQUES

PROGRAMMATION OBJET JAVA



PROFESSEUR : LAHCEN MOUMOUN

Sommaire

Introduction

Java est l'un des langages de programmation les plus utilisés dans le monde, grâce à sa robustesse, sa portabilité et son modèle orienté objet. Conçu pour être simple à apprendre, sécurisé, et extrêmement polyvalent, Java permet de développer des applications de tout type, allant des applications mobiles (Android) aux systèmes embarqués, en passant par les applications web et les grandes solutions d'entreprise.

Ce polycopié a été conçu pour accompagner les étudiants dans l'apprentissage pratique des concepts fondamentaux du langage Java. Grâce à une série progressifs de travaux pratiques, les étudiants seront guidés à travers les bases du langage, tout en découvrant des notions avancées telles que la programmation orientée objet, la gestion des exceptions, et la manipulation des collections. Les travaux pratiques proposés dans ce polycopié permettent, ainsi, à ces étudiants d'approfondir leurs connaissances en appliquant les concepts théoriques du langage JAVA dans des situations pratiques et en en se familiarisant avec des outils et des environnements de développement standards comme l'IDE Eclipse ou IntelliJ IDEA.

TP N°1 : Eléments de base du langage JAVA

Exercice n°1

Les nombres de Amstrong appelés parfois nombres cubes sont des nombres entiers qui ont la particularité d'être égaux à la somme des cubes de leurs chiffres. Par exemple, 153 est un nombre de Amstrong car on a : $153 = 1^3 + 5^3 + 3^3$.

Afficher en java tous les nombres de Amstrong sachant qu'ils sont tous compris entre 100 et 499. Indication : Les nombres de Amstrong sont : 153, 370, 371 et 407.

Exercice n°2

Considérons une série Sm définit par la formule suivante :

$$Sm = \sum_{n=1}^{m} \left(\frac{2^n\, Un}{n!}\right) = 2 * U1 + 2^2 * \frac{U2}{2} + 2^3 * \frac{U3}{3!} + \ldots + 2^m \frac{Um}{m!}$$

Avec Un est une suite définit par la formule de récurrence :

$$\begin{cases} U_1 = 1 \\ U_n = \dfrac{1}{U_{n-1}} + 1 \ \ (n > 1) \end{cases}$$

1. Ecrire une méthode appelée **suite** qui reçoit en entrée un entier n et qui retourne la valeur Un.
2. Ecrire une méthode appelée **fact** qui reçoit en entrée un entier n et qui retourne la valeur n !
3. Ecrire une méthode appelée **série** qui reçoit en entrée un entier m et qui utilise les fonctions précédentes pour retourner la valeur de la série Sm.
4. En utilisant les méthodes précédentes, établir une méthode principale qui répète le calcul de la série Sm (pour des valeurs incrémentales de m) jusqu'à ce que cette série converge vers une valeur finale. On prend une précision de convergence de l'ordre de 0.01 .

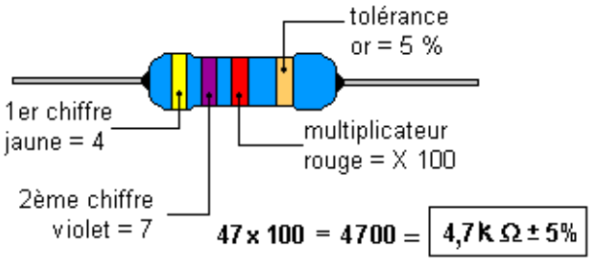
Exercice n°3

Dans l'objectif de mettre en place un certain nombre d'outils de calcul destinés aux électroniciens débutants, on vous propose de faire un programme traitant le code de couleurs des résistances.

Soient les déclarations globales suivantes :

Couleur	Code
Noir	0
Rouge	1
Jaune	2
Bleu	3
Gris	4

Couleur	Code
Brun	5
Orange	6
Vert	7
Violet	8
Blanc	9



1. Ecrire une méthode appelée **dem_coul**, qui ne reçoit aucun paramètre en entrée, qui affiche le nom et le code des dix couleurs ci-dessus, demande à l'utilisateur laquelle il choisit (entier entre 0 et 9) et retourne cet entier.
2. Ecrire une méthode appelée **multiplicateur** qui reçoit en entrée un entier n (entre 0 et 9) et retourne un flottant valant 10^n. Vu la faible valeur de n, on calculera cette puissance en faisant des multiplications par 10 dans une boucle.
3. Etablir une méthode principale, utilisant les méthodes précédentes, demandant à l'utilisateur les trois couleurs correspondant aux trois chiffres significatifs, puis la couleur du multiplicateur, et affichant la valeur de la résistance, en W (Ohms).
4. annales d'examens permettent de se familiariser avec le format des épreuves et de tester des connaissances dans des conditions réelles. En s'exerçant sur ces sujets, les étudiants pourront non seulement mieux comprendre les attentes des examens, mais aussi développer des stratégies pour gérer leur temps et améliorer leur performance.

Elément de correction

Exercice 1

```
public class ProgAmtr {
   public static int sommeChiffre(int n) {
   int s=0;
   while (n!=0) {
         int ch=n%10; n=n/10;//n/=10
         s+=ch*ch*ch; }
   return s;
   }
   public static void main(String[] args) {
     for(int ninit=100;ninit<500;ninit++ ) {
         if (sommeChiffre(ninit)==ninit)
           System.out.println("Le nombre:"+ninit+ " est Amstrong");
     }
   }
}
```

Exercice 2

```
public class test {
   public static float suite (int n) {
         float u=1.F;
         for (int i=2; i<=n;i++)u=1/u+1;
         return u;   }
   public static float fact (int n) {
         float f=1.F;
         for (int i=2; i<=n;i++)f*=i;
         return f;   }
   public static float serie (int m) {
         float sm=0.F;float p2=1.F;
         for (int n=1; n<=m;n++) {
               p2*=2; sm+=(p2*suite(n)/fact(n));  }
   return sm;
   }
   public static void main(String[] args) {
   int m=1;    float sm0,sm1=serie(m++);
   do {  sm0=sm1; sm1=serie(m++);
   } while (Math.abs(sm1-sm0)<0.001);
   System.out.println("sm ="+sm1);
   }
}
```

TP N°2 :Classes et objet en JAVA

1. Un employé est caractérisé par les informations suivantes: un nom, une adresse, un téléphone et un email. Un employé peut être un employé à plein temps ou bien à temps partiel. Il faut également stocker le volume horaire travaillé par semaine ainsi que le montant horaire perçu par l'employé. Dans le cas d'un employé à plein temps, son volume horaire étant fixe à 44 heures. Définir une classe nommée «Employé» en se limitant aux méthodes publiques suivantes:

    - Un constructeur pour créer et initialiser les attributs d'un employé à plein temps et un autre constructeur pour créer un employé à temps partiel. On suppose qu'à chaque employé (plein temps ou à temps partiel) est attribué un numéro d'ordre (1ier , 2ème …) relatif à l'instantiation.
    - « getSalaire » pour retourner le salaire mensuel d'un employé.
    - « toString » pour retourne les informations d'un employé (nom, adresse, téléphone, email , plein temps ou à temps partiel et salaire).

2. On veut maintenant stocker des informations sur les entreprises qui embauchent des employés. une entreprise est caractérisée par un nom, une adresse. on considère que les employés d'une entreprise sont stockés dans un tableau et que le nombre d'employés peut varier d'une entreprise à une autre. Ecrire la classe Entreprise comportant les méthodes publiques suivantes:

    - Un constructeur pour créer et initialiser les attributs d'une entreprise. On suppose que le nombre des employés d'une entreprise ne peut pas dépasser 50 employés.
    - ajoutEmploye(Employe e);
    - getNbEmployes() ; // retourne le nombre d'employés de l'entreprise.
    - getEmploye(int ieme) ; // retourner les informations de l'ème employé.
    - getSalaireGlobal() ; // retourne le salaire total de tous les employés de l'entreprise.
    - toStringPleinTemps(); retourne une chaine de caractère en concaténant les informations des employés qui travail en plein temps.

3. Définir une classe nommée « TestEntreprise » qui contient la méthode main(). Cette méthode assure les tests suivants :

- Créer un employé à temps partiel : JAVA habitant 5 bd Descartes et dont le téléphone est le 0148484040 et l'email java@gmail.com. Il travaille 15 heures par semaine à 50 DH de l'heure.
- Créer un employé à temps plein; JEE qui habite 4 rue de Meaux, à pour téléphone 0140404040, son email est jee@hotmail.fr. Il travaille à 40 DH de l'heure.
- Créer l'entreprise ABC dont l'adresse est 99 impasse de java
- Faire en sorte que les employés JAVA et JEE travaillent pour l'entreprise ABC.
- Afficher les salaires des employés de l'entreprise ABC.

Elément de correction

```
public class Employe  {
      private int numero; private static int nb=0;
      private String  nom,  adresse,  telephone, email;
      private boolean pleinTemps;
      private float volumeHoraire , montantHoraire;
      public Employe(String nom, String adresse, String telephone,
            String email, float volumeHoraire, float montantHoraire) {
            this.numero=++nb; this.nom = nom;
            this.adresse = adresse; this.telephone = telephone;
            this.email = email;this.volumeHoraire = volumeHoraire;
            this.montantHoraire = montantHoraire;    }
      public Employe(String nom, String adresse, String telephone,
            String email, float montantHoraire) {
            this.numero=++nb;
            this.nom = nom; this.adresse = adresse;
            this.telephone = telephone;  this.email = email;
            this.pleinTemps=true; this.volumeHoraire = 44;
            this.montantHoraire = montantHoraire; }
      public int getNumero() { return numero;}
      public void setNumero(int numero) { this.numero = numero; }
      public String getNom() { return nom; }
      public void setNom(String nom) { this.nom = nom; }
      public String getAdresse() { return adresse; }
      public void setAdresse(String adresse){this.adresse = adresse;}
      public String getTelephone() { return telephone; }
      public void setTelephone(String telephone){
            this.telephone = telephone; }
      public String getEmail() { return email; }
      public void setEmail(String email) { this.email = email; }
      public boolean isPleinTemps() { return pleinTemps; }
      public void setPleinTemps(boolean pleinTemps) {
            this.pleinTemps = pleinTemps; }
      public float getVolumeHoraire() { return volumeHoraire; }
      public void setVolumeHoraire(float volumeHoraire) {
            this.volumeHoraire = volumeHoraire; }
      public float getMontantHoraire() { return montantHoraire; }
      public void setMontantHoraire(float montantHoraire) {
            this.montantHoraire = montantHoraire; }
      @Override
      public String toString() {
            return "Employe [numero=" + numero + ", nom=" + nom + ",
                  adresse=" + adresse + ", telephone=" + telephone
                  + ", email=" + email + ", pleinTemps=" + pleinTemps +
                  ", volumeHoraire=" + volumeHoraire+ ",
                  montantHoraire=" + montantHoraire + "]"; }
      public float getSalaire() {return montantHoraire*volumeHoraire*4 }
}
```

```
public class Entreprise {
	private String nom, adresse;
	private int nbEmployes;
	private Employe listEmployes[];
	public Entreprise(String nom, String adresse) {
		this.nom = nom;
		this.adresse = adresse;
		this.listEmployes= new Employe[50];
	}
	public void ajoutEmploye(Employe e) {
		listEmployes[nbEmployes++]=e; }
	public int getNbEmployes() { return nbEmployes;	}
	public Employe getEmploye(int ieme) {
		return listEmployes[ieme];	}
	public float getSalaireGlobal() {
		float sT=0.F;
		for (int i=0; i<nbEmployes;i++) sT+=listEmployes[i].getSalaire();
		return sT;
	}
	public String toStringPleinTemps() {
		String str="Les employes en plein Temps:\n";
		for (int i=0;i<nbEmployes; i++)
			if (listEmployes[i].isPleinTemps())
				str+=listEmployes[i].toString()+"\n";
		return str;
	};
}
public class TestEntreprise {
	public static void main(String[] args) {
		Employe e1=new Employe("JAVA", "5 bd Descartes",
			"0148484040", "java@gmail.com", 15.F,50.F);
		Employe e2=new Employe("JEE", "4 rue de Meaux",
			 "0140404040", "jee@hotmail.fr", 40.F);
		Entreprise en=new Entreprise("ABC", "99 impasse de java");
		en.ajoutEmploye(e1);en.ajoutEmploye(e2);
		for (int i=0; i<en.getNbEmployes();i++)
			System.out.println(en.getEmploye(i));

	}
}
```

TP N°3 : Classes et objets membres

On suppose que le fond documentaire d'une bibliothèque soit compose uniquement d'ouvrage (c.à.d. de livres). Le but de cet exercice de modéliser les ouvrages de cette bibliothèque.

1. Définir une classe nommée «Auteur» permettant de modéliser les auteures. Elle possède:

- Deux attributs privés de type «String», à savoir: le nom et le prénom de l'auteur.
- Un constructeur pour créer et initialiser tous les attributs d'un objet de type «Auteur».
- Des accesseurs « getNom» et « getPrenom » permettant de retourner, respectivement, le nom et le prénom de l'auteur.
- La méthode publique « toString » permettant de retourner la chaine:

nomAuteur + "-- " + prenomAuteur

2. Définir une classe nommée «Ouvrage» pour modéliser les ouvrages. Elle possède:

- Au moins trois attributs privés, à savoir :
  - code (type String) pour identifier de manière unique chaque ouvrage. Le code est composé d'un numéro qui correspond à l'ordre de création de l'ouvrage suivi des trois premières lettres du titre de l'ouvrage.
  - titre (type String) qui mémorise le litre de l'ouvrage
  - auteur (type classe Auteur) pour mémoriser le nom et le prénom du premier auteur.
  - estEmprunte(type boolean) qui indique si l'ouvrage est emprunté (valeur true) ou non (valeur false).
- Un constructeur pour créer et initialiser tous les attributs d'un objet de type « Ouvrage ».

- Au moins les méthodes publiques suivantes (en cas de besoin, vous rajoutez d'autres méthodes):
  - Les accesseurs « getCode », « getTitre », « getAuteur » et « getEstEmprunt » pour retourner respectivement le code , le titre, l'auteur et l'état d'emprunt de l'ouvrage.
  - « toString » pour retourne la chaine :
  - code + "-- " titre + "-- "+ nomAuteur + "-- " + prenomAuteur
  - « Emprunte » et « retour » pour affecter à l'attribut « estEmprunte » , recpectivement, la valeur « true » et « false ». la méthode « Emprunte » returne « false» si l'ouvrage est déjà emprunté.

3. Définir une classe nommée « Bibliotheque » pour modéliser la bibliothèque. Elle possède :

- Au moins quatre attributs privés, à savoir :
  - Les entiers nbAuteurs et nbOuvrages qui mémorisent, respectivement, le nombre des auteurs et ceux des ouvrages de la bibliothèque.
  - lesAuteurs (type tableau d'Auteur) pour mémoriser les auteurs de la bibliothèque.
  - lesOuvrages (type tableau d'Ouvrage) pour mémoriser tous ouvrages de la bibliothèque.
- Un constructeur pour créer et initialiser tous les attributs d'un objet de type «Bibliotheque». On suppose que la taille maximale des tableaux est saisie au clavier.
- Au moins les méthodes publiques suivantes (en cas de besoin, vous rajoutez d'autres méthodes) :
  - ajouteAuteurs(String nomAuteur, String prenomAuteur); // Ajoute d'un auteur.

- ajouteOuvrage(String titre, Auteur auteur); // Ajoute un ouvrage à la bibliothèque.
- afficheEmpruntes() ; // Affiche tous les ouvrages empruntés.
- emprunte(String code); // emprunter l'ouvrage dont le code est passé en paramètre. La méthode retourne « false» si l'ouvrage est déjà emprunté

4. Définir une classe nommée « TestBibliotheque » qui contient la méthode main(). Cette méthode assure les tests suivants :

- Créer une bibliothèque (un objet de type Bibliothèque) composée de 4 ouvrages.
- Emprunter le deuxième et le troisième ouvrage.
- Afficher tous les ouvrages empruntés.

TP N°4 : Héritage, polymorphisme et exceptions

On désire une application de vente et d'achat de produits(livre, CD, …) d'un magasin.

A. Nous considérons que chaque produit est caractérisé par son numéro, son nom, son prix d'achat, son prix de vente, le nombre d'exemplaires en stock.

A la création du produit, on fixe son nom, son prix d'achat et son prix de vente, c'est à-dire que ces données sont fournies en argument au constructeur de la classe. Le numéro d'un produit est généré automatiquement, il correspond au numéro d'ordre de l'instanciation de la classe « Produit ». La classe Produit dispose d'un certain nombre de méthodes, qui lui permettent d'augmenter ou de diminuer le nombre d'exemplaires en stock, ainsi que de modifier (setter) et d'obtenir les valeurs des différents attributs (le numéro d'un produit ne peut pas être modifié).

Ecrivez la classe Produit.

B. Un magasin se caractérise par son nom et par son stock de produits. Le stock de produit est représenté par un tableau « listProduit » d'objets « Produit ».

Avant de pouvoir acheter ou vendre un produit, il faut l'avoir ajouté dans le stock. Pour cela, la classe Magasin dispose d'une méthode :

public void ajouterProduit(String nom, float prixAchat, float prixVente)

Pour acheter ou vendre ce produit, on utilise alors le numéro comme argument des méthodes:

- public void acheterProduit(int numeroProduit, int nombreExemplaires)
- public void vendreProduit(int numeroProduit, int nombreExemplaires)

La classe Magasin dispose également des méthodes habituelles d'accès à ses attributs. La méthode « **vendreProduit** » lève l'exception « **produitNonDisponible** » si le nombre d'exemplaires en stock est égale à 0.

Réaliser la classe Magasin.

C. Supposons maintenant qu'on veut prendre en considération les différents types de produits. Par exemple, on aimerait avoir une classe Livre qui ait le même

comportement que la classe Produit, mais qui dispose d'un attribut auteur et d'un attribut éditeur.

Ecrivez la classe Livre qui hérite de la classe Produit.

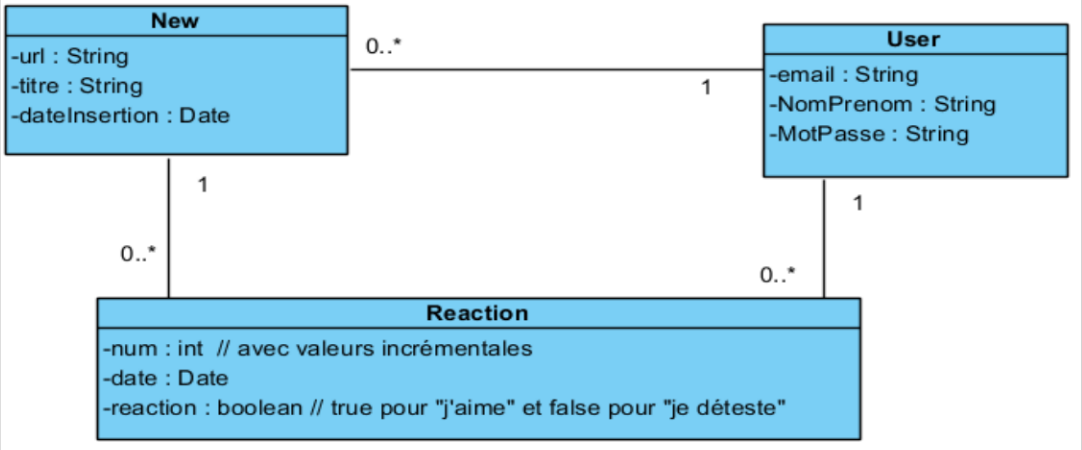
D. Proposer un programme de test permettant d'instancier un magasin, de lui ajouter 2 livres, d'acheter 10 exemplaires du premier livre et 20 exemplaires du 2ème livre puis de vendre 2 exemplaires du premier livre (maitre en évidence le concept de polymorphisme)

TP N°5 : Classes et associations en JAVA.

Notre objectif à travers ce TP est le développement d'une application de gestion des news. Une telle application est basée sur le principe suivant :

- Un utilisateur doit s'authentifier pour accéder aux différents news. Les nouveaux utilisateurs peuvent s'inscrire en fournissant leur email considéré comme identifiant, le nom + prénom et le mot de passe.
- Un utilisateur peut ajouter une news au site si elle n'existe pas déjà dans le site, c'est à dire référencer une page Web (une URL) pour signaler son existence aux autres utilisateurs.
- Un utilisateur peut aimer ou détester un news ajouté précédemment par un utilisateur.

Nous considérons que l'analyse qui a été effectuée pour cet application a permis d'élaborer le diagramme suivant :



A. Dans l'implémentation JAVA, nous ajoutons à la classe « Reaction » un objet membre user et un objet membre news qui représentent respectivement l'utilisateur qui a réagi et la news associée à une réaction. Également, un objet membre user est ajouté à la classe « News » pour sauvegarder l'utilisateur qui a créé la news. Un conteneur d'objet static est ajouté à la classe « User » pour regrouper (avec une insertion automatique lors de l'instanciation) l'ensemble des utilisateurs. Les classes métiers correspondantes au diagramme ci-dessus sont les suivantes :

User
- email : String - nomPrenom :String - motPasse : String - users : User [ ]
+ User (…) + getXxx():…; + setXxx(…):… + equals(User other) : boolean

News
- url : String - titre : String - dateNews : Date // Date courante - createur : User
+ News (…) + getXxx(): … ; + setXxx(…): … + equals(News other) : boolean

Reaction
- num // numéro avec valeurs incrémentales - dateReaction : Date // Date courante - reaction : boolean // true pour "j'aime" et false pour "je déteste" - user : User - news : News
+ Reaction (…) + getXxx(): … ; + setXxx(…): … // Xxx represente un attribute + equals(Reaction other) : boolean

B. Nous considérons que les différents services métiers de notre application sont définis dans la classe « ServiceMetier » suivante :

ServiceMetier
- listNews : static News [ ] - listReactions : static Reaction [ ]
+ ServiceMetier () + getXxx(): … ; + setXxx(…) : … + getUser(email: String): User + getUser(email: String, motPasse:String) + getNewsByUrl(url: String): News + getAllUrl():ArrayList<String> + addNews (news: News) : boolean + addReaction (reaction: Reaction) : boolean + getNewsCreateByUser(user: User): ArrayList<News> + getReactionByUser(user: User): ArrayList<Reaction> + getReactionByNews(news: News): ArrayList<Reaction> + getNombreJaime(news: News): int + getNombreJeDeteste(news: News): int

Travail demandé :

1. Ecrire les différentes classes de l'application
2. Réaliser une classe pour tester les différentes fonctionnalités de l'application.

Éléments de correction

```
public class User {
   private String email, nomPr, motPass;
   private static User [] users;
   private static int nb;
   static  {  users=new User[100];  nb=0; }
   public User(String email, String nomPr, String motPass) {
         this.email = email;     this.nomPr = nomPr;
         this.motPass = motPass;  users[nb++]=this;     }
   public static User[] getUsers() {
         User usersTemp[]=new User[nb];
         for (int i=0; i<nb; i++)
   usersTemp[i]=users[i];
         return usersTemp;
   }
    public static User getUser(String email) {
         for (int i=0; i<nb; i++)
               if (users[i].email.equals(email))  return users[i];
         return null;   }
    public static User getUser(String email, String PassWord) {
         for (int i=0; i<nb; i++)
               if (users[i].email.equals(email) &&
                     users[i].motPass.equals(PassWord)) return users[i];
         return null;
    }
   public String getEmail() { return email;  }
   public void setEmail(String email) {     this.email = email; }
   public String getNomPr() {    return nomPr;     }
   public void setNomPr(String nomPr) { this.nomPr = nomPr;   }
   public String getMotPass() {  return motPass;}
   public void setMotPass(String motPass) {  this.motPass=motPass;}
   @Override
   public boolean equals(Object obj) {
         if (this == obj)  return true;
         if (obj == null)  return false;
         if (getClass() != obj.getClass())  return false;
         User other = (User) obj;
         return Objects.equals(email, other.email);  }
   @Override
   public String toString() {
         return "User [email=" + email + ", nomPr=" +
               nomPr + ", motPass=" + motPass + "]";    }
}
```

```
public class News {
   private static SimpleDateFormat formatter =
            new SimpleDateFormat ("dd-MM-yyyy", Locale.FRANCE);
   private String url, titre, dateInsertion;
   private User createur;
   public News() {   dateInsertion =new Date();    }
   public News(String url, String titre, User createur) {
         this();//ou dateInsertion=new Date();
         this.url = url;   this.titre = titre;
         this.createur = createur;     }
   public String getDateNewsString() {
         return (String) formatter.format (dateInsertion);}
   public String getUrl() { return url;      }
   public void setUrl(String url) {this.url = url; }
   public String getTitre() {    return titre;     }
   public void setTitre(String titre) { this.titre = titre;   }
   public Date getDateInsertion() { return dateInsertion; }
   public void setDateInsertion(Date dateNews) {
         this.dateInsertion = dateNews;      }
   public User getCreateur() {   return createur;  }
   public void setCreateur(User createur) {this.createur = createur;}
   @Override
   public String toString() {
   return "News [url=" + url + ", titre=" + titre +
               ", dateNews=" + getDateNewsString() +
               ", createur=" + createur.getNomPr() + "]";
   }
   @Override
   public boolean equals(Object obj) {
         if (this == obj) return true;
         if (obj == null)  return false;
         if (getClass() != obj.getClass())  return false;
         News other = (News) obj;
         if (url == null) { if (other.url != null) return false;}
         else if (!url.equals(other.url))   return false;
         return true;
   }
}
```

```
public class Reaction {
   private static SimpleDateFormat formatter =
            new SimpleDateFormat ("dd-MM-yyyy", Locale.FRANCE);
    private int num;  private Date dateReaction;
   private boolean reaction; private User user; private News news;
   private static int nb=0;
   public Reaction() {dateReaction =new Date();   nb++; num=nb; }
   public Reaction( boolean reaction, User user, News news) {
         this();this.reaction = reaction;
         this.user = user; this.news = news;}
   public String getDateReactionString()    {
         return (String) formatter.format (dateReaction.getTime());}
    public Date getDateReaction() {return dateReaction;  }
   public void setDateReaction(Date dateReaction) {
         this.dateReaction = dateReaction;  }
   public boolean getReaction() {  return reaction;}
   public String getReactionString() {
         if(reaction) return ("J'aime"); else return ("je Deteste");
   }
   public void setReaction(boolean reaction) {
          this.reaction = reaction;   }
   public User getUser() { return user;}
   public void setUser(User user) {   this.user = user; }
   public News getNews() { return news;      }
   public void setNews(News news) {   this.news = news; }
    public int getNum() {   return num; }
   @Override
   public String toString() {
    return "Reaction [num=" + num + ", dateReaction="
        + getDateReactionString() + ", reaction=" + reaction + ", "
               + "user=" + user.getNomPr()
               + ", news=" + news.getTitre() + "]";
   }
   @Override
   public boolean equals(Object obj) {
         if (this == obj)  return true;
         if (obj == null)  return false;
         if (getClass() != obj.getClass())  return false;
         Reaction other = (Reaction) obj;
         return this.news.equals(other.news)
               && this.user.equals(other.user);
   }
}
```

```
public class ServiceMetier {
   private static  News [] listNews = new News[100]  ;
   private static  Reaction listReactions[]= new Reaction[10000] ;
   private static int nbNews;    private static int nbReaction;
   public static News[] getListNews() {
         News newsTemp[]=new News[nbNews];
         for (int i=0; i<nbNews; i++)
               newsTemp[i]=listNews[i];
         return newsTemp;
   }
   public static Reaction[] getListReactions() {
         Reaction reactionTemp[]=new Reaction[nbReaction];
         for (int i=0; i<nbReaction; i++)
               reactionTemp[i]=listReactions[i];
         return reactionTemp;
   }
   public static User getUser(String email) {
         return User.getUser(email);
   }
   public static User getUser(String email, String PassWord) {
         return User.getUser(email, PassWord);
   }
   public static News getNewsByUrl(String url) {
         for (  News news : listNews  ) {
               if (news==null) continue;
               if ( news.getUrl().equals(url)  ) return news;
         }
         return null;
   }
   public static boolean addNews(News news) {
         if (getNewsByUrl(news.getUrl())!=null)return false;
         listNews[nbNews++]=news;
         return true;      }
   public static int getReation (User user, News news) {
         for ( int i = 0; i< nbReaction; i++ )
               if (listReactions[i].getUser().equals(user) &&
                     listReactions[i].getNews().equals(news))
                     return i;
         return -1;
}
```

```
    public static boolean addReaction(Reaction reaction) {
          if (  getNewsByUrl( reaction.getNews().getUrl() )== null  ||
                reaction.getUser()== null) return false;
          int pos=getReation(reaction.getUser(), reaction.getNews());
          if (pos==-1) listReactions[nbReaction++]=reaction;
          else  listReactions[pos]=reaction;
          return true;
    }
}
public class Test {
    public static void main(String[] args) {
    for (int i=1; i<=5; i++)
          new User("ensab"+i+"@ac.ma","ensab"+i, "ensab"+i);
    for(User0 user : User0.getUsers()) System.out.println(user);
    for (int i=0; i<5; i++)
          ServiceMetier.addNews(
                new News("url " + (i+1) , "Tritre " + (i+1),
                                  User.getUsers()[i] )    );

    for(News news : ServiceMetier.getListNews())
          System.out.println(news);
    System.out.println(ServiceMetier.getUserByEmail("ensab3@ac.ma"));
    boolean react=false;
    for ( News news : ServiceMetier.getListNews() ) {
          for (User user : User.getUsers()) {
            if (news.getCreateur().equals(user)) continue;
            react=!react;
            ServiceMetier.addReaction(new Reaction(react, user, news));
          }
    }
    for (Reaction reaction : ServiceMetier.getListReactions()) {
          System.out.println(reaction); }
}
}
```

TP N°6 : Classes abstraites, interfaces et exceptions

Les organisateurs des jeux olympiques souhaitent informatiser la gestion de cet événement. Dans ce sujet on s'intéresse en particulier aux épreuves individuelles et à la gestion de leurs résultats.

A. **Athlètes** Les jeux olympiques réunissent des athlètes issus de très nombreux pays (la liste de ces pays est fixée et connue à l'avance). Les athlètes sont identifiés par leur nom, leur prénom et le pays pour lequel ils concourrent.

   Donnez le code java d'une classe Athlete pour laquelle on disposera des méthodes accesseurs, d'une méthode toString reprenant les informations et une méthode equals deux athlètes étant égaux s'ils ont même nom, prénom et nationalité.

B. **Résultats** Les résultats des athlètes sont représentés par des objets du type Resultat :

Resultat <Abstract Class>
- athlete : Athlete - resultat : Object
+ toString() : String + compareTo(Resultat r) : int

   Le constructeur de la classe « Resultat » recoit uniquement la référence de l'athlète. La valeur du resultats sera fixé par un setter. Selon les épreuves la nature du résultat change. Il peut se mesurer :

   - par un temps, c'est par exemple le cas des courses, une telle donnée est modélisée par une instance de la classe **ResultatTemps** ;
   - par une distance, c'est par exemple le cas des lancers ou des sauts, une telle donnée est modélisée par une instance de la classe **ResultatDistance** ;
   - par un nombre de points, comme c'est le cas en plongeon, en tir ou au décathlon, une telle donnée est modélisée par une instance de la classe **ResultatPoints** ;
   - par un rang dans un classement, comme c'est le cas au tennis de table ou pour les sports de combat, une telle donnée est modélisée par une instance de la classe **ResultatRang**.

Les résultats de même nature sont comparables deux à deux (une exception de ClassCastException est levée dans le cas contraire) , les classes ci-dessus implémentent donc également l'interface **Comparable** (cf. page 2/2). La méthode **compareTo** qui définit la relation d'ordre permet alors de déterminer si un résultat est ou non meilleur qu'un autre.

1. Donnez un code java pour la classe ResultatDistance. Un tel résultat est meilleur qu'un autre si sa distance est plus grande.
2. Donnez un code java pour la classe ResultatRang. Un tel résultat est meilleur qu'un autre si son rang est inférieur.

C. **Epreuves**. Les épreuves sont modélisées par des instances de la classe **Epreuve**. Ces objets permettent de gérer les informations concernant les résultats des athlètes participant à l'épreuve. Ces informations sont stockées dans un tableau.

Une épreuve peut-être ou non déclarée close (ou terminée). Lorsque c'est le cas les résultats sont définitifs.

Les méthodes accessibles pour des épreuves permettent :

- Méthode « toString » pour connaitre sa dénomination (”100m haies féminin”, ”Tennis de table masculin”, etc.),
- Méthode « ajouteAthlete » pour l'ajout d'une instance resultat du participant à l'épreuve. Une exception illegalResult sera lévée si le type du resultat est différents à ceux de la collection des résultats.
- Méthode « estClose » pour savoir si une épreuve est terminée ou non,
- Méthode « cloture » pour la cloture d'une épreuve,
- Méthode « getRecordOlymique » pour connaître le record olympique en vigueur avant le déroulement de l'épreuve, cette donnée est de type Resultat,
- Méthode « fixeResultat » pour l'ajout ou la mise à jour du résultat d'un participant de l'épreuve, rien ne se passe si le participant n'était pas inscrit à l'épreuve, une exception IllegalStateException est déclenchée si l'épreuve a été déclarée close,
- Méthode « getResultat » pour connaître le résultat d'un athlète à l'épreuve, la valeur null est renvoyée si l'athlète ne participe pas à l'épreuve ou si aucun résultat n'a été ajouté pour lui,

A la construction d'une épreuve, seule sa dénomination et le record olympique en vigueur (pour les épreuves dont le résultat est de type rang, cette notion n'a pas réellement de sens, sa aleur sera alors mise à null) sont communiqués.

1. Donnez un code java pour la classe Epreuve satisfaisant le cahier des charges ci-dessus.
2. On souhaite ajouter quelques fonctionnalités à la classe Epreuve afin de gérer les médaillés. Le vainqueur d'une épreuve est celui qui a obtenu le meilleur résultat. La médaille d'or lui est alors décernée. Donnez le code java d'une méthode getMedailleOr qui a pour résultat l'athlete vainqueur de l'épreuve une fois qu'elle est déclarée close, dans le cas contraire une exception IllegalStateException est levée.

Pour la suite on supposera que des méthodes similaires getMedailleArgent et getMedailleBronze ont été définies.

D. **Les Jeux**. Les jeux olympiques sont gérés par la classe **JeuxOlympique** dont chaque instance est caractérisée par l'année du déroulement de ces jeux (un int), la ville organisatrice (un String) et la liste des objets Epreuve qui la composent (Tableau de type de base « Epreuve »).

1. Donnez un code java pour la classe Epreuve satisfaisant le cahier des charges ci-dessus.
2. Donnez le code java d'une méthode nbRecordsBattus qui indique le nombre d'épreuves terminées pour lesquelles le record olympique a été battu par le vainqueur au cours de l'épreuve.
3. Un classement entre les pays participant est réalisé. Pour cela on attribue un nombre de points pour chaque médaille d'or, d'argent ou de bronze, par exemple 10, 3 et 1. Ces valeurs sont définies par des constantes dans la classe JeuxOlympiques. Donnez le code java de la méthode getScore correspondante

Éléments de correction

```
public class Athlete {
   private String nom, prenom, pays;
   private static ArrayList<String> listPays;
   public static ArrayList<String> getListPays() { return listPays;}
   public static void addPays(String pays) { listPays.add(pays);}
   static {
         listPays= new ArrayList<String>();
         listPays.add("Maroc");listPays.add("France");
         listPays.add("Espagne"); }
   public Athlete(String nom, String prenom, String pays) {
         this.nom = nom;   this.prenom = prenom; this.pays = pays; }
   @Override
   public String toString() {
      return "Athlete [nom=" + nom + ", prenom=" + prenom + ", pays="
         + pays + "]";
   }
   public String getNom() { return nom;      }
   public void setNom(String nom) {    this.nom = nom;   }
   public String getPrenom() {   return prenom;    }
   public void setPrenom(String prenom) {this.prenom = prenom;}
   public String getPays() {return pays;}
   public void setPays(String pays) {  this.pays = pays; }
   @Override
   public boolean equals(Object obj) {
         if (this == obj) return true;  if (obj == null) return false;
         if (getClass() != obj.getClass()) return false;
         Athlete other = (Athlete) obj;
         if (nom == null) { if (other.nom != null) return false;
         } else if (!nom.equals(other.nom)) return false;
         if (pays == null) {
         if (other.pays != null) return false;
         } else if (!pays.equals(other.pays)) return false;
         if (prenom==null){if (other.prenom != null)return false;
   } else if (!prenom.equals(other.prenom)) return false;
         return true; }
}
public class Epreuve {
   private String denomination;  private Object recordOlymique;
   private Boolean close;
   private ArrayList<Resultat> resultats;
```

```
    public Epreuve(String denomination, Object recordOlymique) {
          this.denomination = denomination;
          this.recordOlymique = recordOlymique;
          resultats= new ArrayList<Resultat>(); }
    public String getDenomination() {   return denomination;    }
    public void setDenomination(String denomination) {
          this.denomination = denomination;  }
    public Object getRecordOlymique() { return recordOlymique;  }
    public void setRecordOlymique(Object recordOlymique) {
          this.recordOlymique = recordOlymique;}
    public Boolean estClose() {   return close;}
    public void cloture () {this.close = true;}
    public ArrayList<Resultat> getResultats() {return resultat }
    @Override
    public String toString() {
          return "Epreuve [denomination=" + denomination + "]" }
    public void ajouteAthlete (Resultat resultat) {
          resultats.add(resultat);}
    public void fixeResultat( Athlete athlete, Object result) {
          for(int i=0; i< resultats.size(); i++) {
                if (resultats.get(i).getAthlete().equals(athlete)) {
                      resultats.get(i).setDist(result); break;}
    } }
}
public abstract class  Resultat implements Comparable<Resultat>{
    private Athlete athlete; private Object dist;
    public Resultat(Athlete athlete) {
          super(); this.athlete = athlete;    }
    public Athlete getAthlete() { return athlete;  }
    public void setAthlete(Athlete athlete) { this.athlete = athlete; }
    public Object getDist() { return dist;}
    public void setDist(Object dist) {  this.dist = dist; }
    public abstract String toString() ;
    @Override
    public abstract int compareTo(Resultat r);
}
public class ResultatDistance extends Resultat{
    public ResultatDistance(Athlete athlete, float dist) {
          super(athlete, dist);   }
    @Override
    public String toString() {
          return "ResultatDistance [Athlete =" + getAthlete() + ",
          Dist=" + getDist() + "]"; }
```

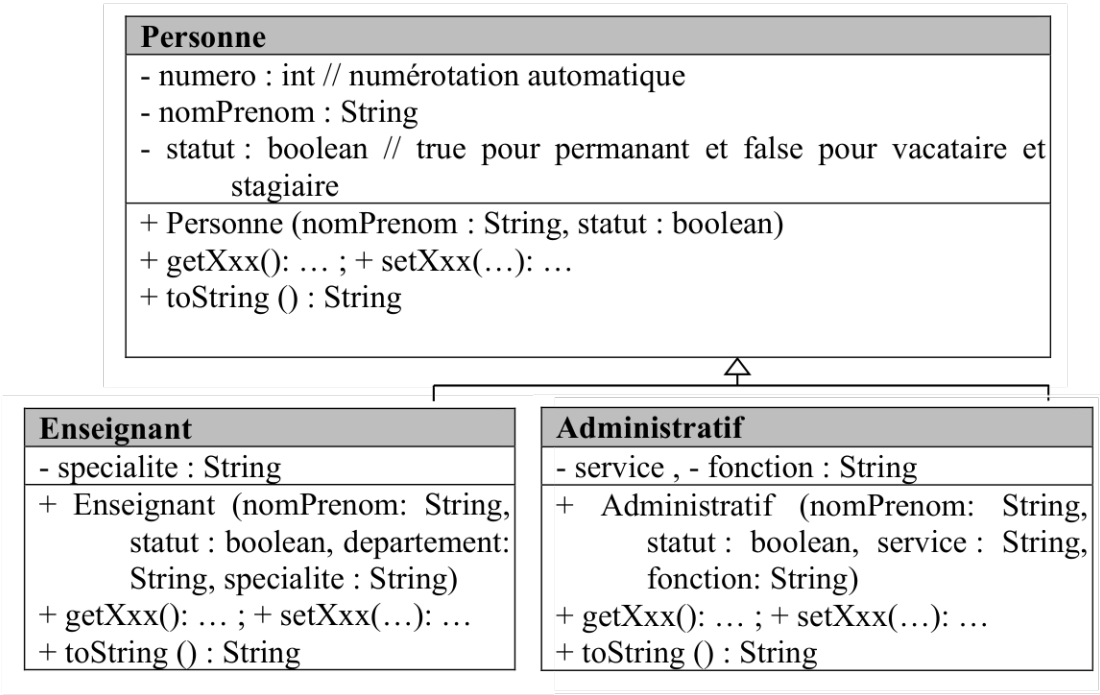
```
   @Override
   public int compareTo(Resultat r)throws ClassCastException   {
         if (this.getClass()!=r.getClass())
               throw new ClassCastException();
         return ((int)getDist()) -
         (int) ( (ResultatDistance)r ).getDist();
   }
}
public class Test {
   public static void main(String[] args) {
     for (int i=1; i<=10; i++) {
        String pays=Athlete.getListPays().get(0);
         Athlete at1= new Athlete("Nom "+i,"Prenom "+i,pays);
         System.out.println(at1.toString());}
}

}
```

TP N°7 : Gestion des Exceptions et Collections

On désire réaliser un programme de gestion du personnel d'un groupe de centre de formation.

A. Considérons une classe nommée «**Personne**» permettant de modéliser les caractéristiques et le comportement d'une personne. Afin de distinguer les enseignants et les administratifs du centre, nous proposons d'utiliser les deux classes «**Enseignant**» et «**Administratif**». La représentation graphique ci-dessous illustre la hiérarchie et la définition de ces trois classes.



1. Donner le code du constructeur de la classe «**Personne**»
2. Définir le constructeur de la classe «**Enseignant**»

B. Afin de gérer le personnel d'un centre de formation, nous Considérons une classe «**Centre**» dont la définition est donnée par la représentation graphique ci-dessous. L'ajout d'une nouvelle personne non permanente (vacataire ou stagiaire) est conditionné par le faite que le nombre total des non permanents ne doit pas être supérieur à 40% du nombre des personnes du centre. Ainsi, une classe nommée « **DepacementNbnNonPermanant** » est associée à l'exception qui sera levée lorsque

l'ajout d'une personne non permanente provoque le déplacement du nombre de non permanant autorisé.

Centre
- intitule , adresse : String // nom et adresse du centre - listPersonnes : ArrayList<Personne> // collection regroupant les enseignants et les administratifs
+ Centre (intitule : String, adresse : String) + getIntitule(), getAdresse() : String + getNbPersonnes() : int // retourne le nombre total des personnes (enseignants et administratifs) + getPersonne(i :int) : Personne // retourne l'objet Personne d'indice i (enseignant ou administratif) + getPersonne(statut : boolean) : ArrayList<Personne> // retourne une liste de personne ayant un statut donné + getEnseignantSpecialite(specialite : String) : ArrayList<Personne> + addEnseignant(Enseignant ens) : void // ajout d'un enseignant à la liste listPersonnes + addAdministratif (Administratif admin) : void // ajout d'un administratif à la liste listPersonnes

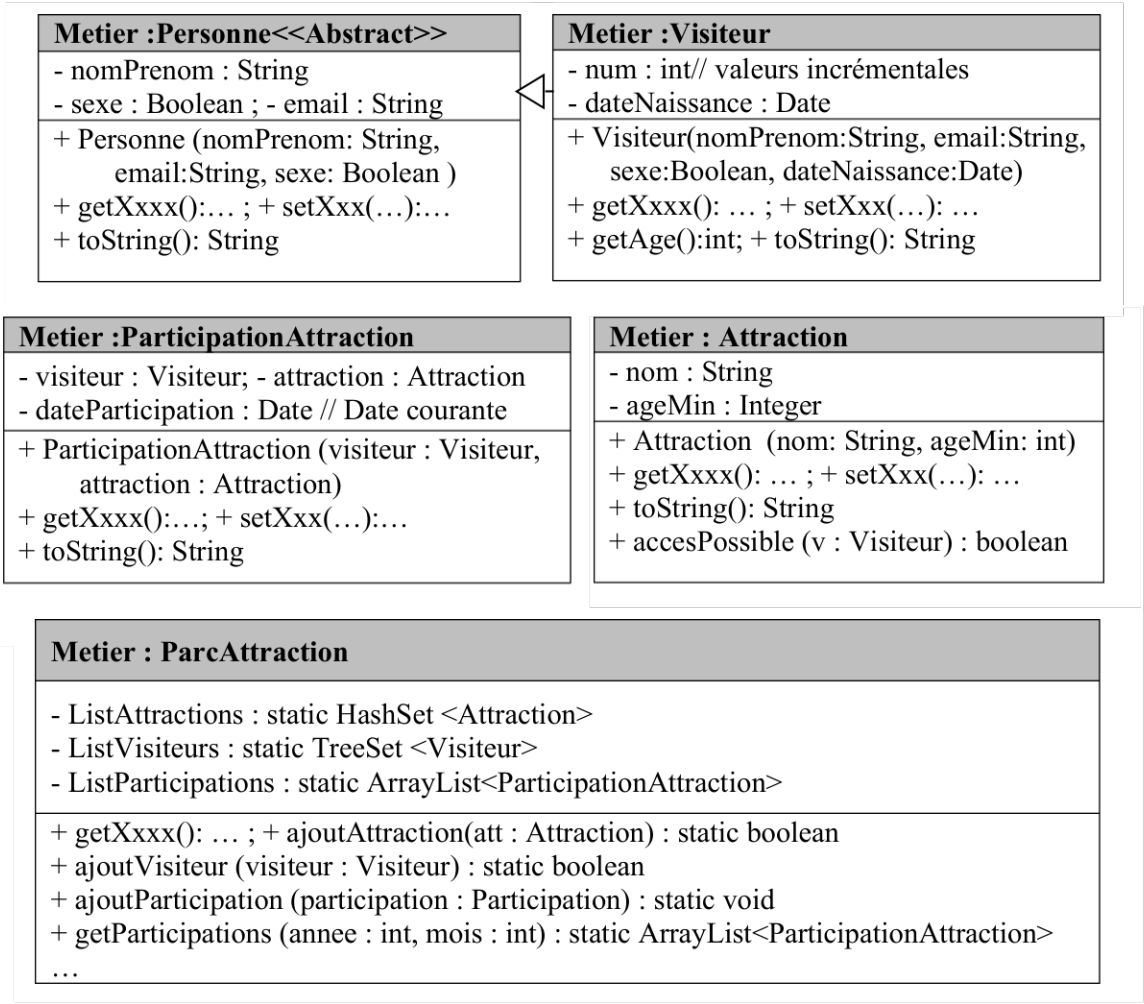
1. Donner le code du constructeur de la classe « **Centre**» **.**
2. Écrire la méthode « **addEnseignant** » permettant d'assurer l'ajout d'un nouvel enseignant à la liste des personnes du centre (« listPersonnes »)**.**
3. Écrire la méthode « **getEnseignantSpecialite**» permettant de retourner la liste des enseignant ayant la spécialité fournie en argument **.**

C. Définir une classe nommée « **TestGestCentre** » qui contient la méthode « main ». Cette méthode assure les tests suivants : Créer un objet de type « **Centre** » et ajouter à la liste de ses personnes un enseignant permanant, un enseignant vactaire, un administratif permanant et un administratif stagiaire;

TP N°8 : Les collections en JAVA

On s'intéresse dans ce sujet au développement d'une application Web de gestion d'un parc d'attractions qui propose à ses visiteurs une liste d'attractions (Jeux et loisirs). L'accès à ces attractions peut être contraint au respect d'une condition d'âge.

Nous considérons la couche métier de l'application. Les visiteurs du parc sont, ainsi, modélisés par une classe « Visiteur » dérivé de la classe «Personne». Les attractions sont modélisées par la classe «Attraction» : Une attraction est définie par un nom, et l'âge minimum (condition d'accès). Une instance de la classe «ParticipationAttraction» est créé pour chaque participation d'un visiteur à une attraction, elle ajoutée automatiquement à la liste «ListParticipations» de la classe «ParcAttraction».



- La classe « ParcAttraction » est un conteneur qui regroupe une liste d'attractions du parc, une liste de ses visiteurs et une liste des différentes participations qui leurs sont associées (tous les attributs et méthodes de la classe sont static).
- Le constructeur de la classe es « ParticipationAttraction » lève une exception « AccesInterditException » si la condition d'accès relative à l'âge n'est pas respectée (la méthode « accesPossible » de la classe « Attraction» retourne true si cette condition est respectée, nous considérons que l'exception « AccesInterditException» est définie).
- La méthode « getParticipations » retourne la liste des participations ayant lieu pendant une année et un moi données (les méthodes « getYear () » et « getMonth () » d'un objet « java. util.Date » permettent de déterminer le mois et l'année lui correspondant).

Définir les différentes classes de la couche métier et proposer un programme pour tester leurs fonctionnalités

Éléments de correction

```
public abstract class Personne {
	protected String nomPrenom ;
	protected Boolean sexe;  // true pour masculine
	protected String email ;
	public Personne(String nomPrenom, Boolean sexe, String email) {
		this.nomPrenom = nomPrenom;
		this.sexe = sexe; this.email = email;	}
	public String getNomPrenom() {return nomPrenom;	}
	public void setNomPrenom(String nomPrenom) {
			this.nomPrenom = nomPrenom;  }
	public Boolean getSexe() {   return sexe;}
	public void setSexe(Boolean sexe) {this.sexe = sexe; }
	public String getEmail() {   return email;	}
	public void setEmail(String email) {this.email = email;}
	@Override
	public String toString() {
		return "Personne [nomPrenom=" + nomPrenom + ", sexe=" + sexe
			+ ", email=" + email + "]";  }
	@Override
	public boolean equals(Object obj) {
		if (this == obj)  return true;
		if (obj == null)  return false;
		if (getClass() != obj.getClass())  return false;
		Personne other = (Personne) obj;
		return Objects.equals(email, other.email);
	}
}
public class Visiteur extends Personne {
	private int num ; // num�ro avec valeurs incr�mentales
	private static int nbV;
	private int age ;
	private int taille ;  // un nombre en centim�tres
	private TreeSet<Attraction> listeAttractionsVisiteur ;
	public Visiteur( String nomPrenom, Boolean sexe,
			String email, int age, int taille) {
		super(nomPrenom, sexe, email);	this.num = ++nbV;
		this.age = age;  this.taille = taille;
		listeAttractionsVisiteur= new TreeSet<Attraction>();
	}
	public int getAge() {return age;   }
	public void setAge(int age) {this.age = age;}
	public int getTaille() {return taille;   }
```

```
	public void setTaille(int taille) {this.taille = taille;   }
	public int getNum() {   return num; }
	public TreeSet<Attraction> getListeAttractionsVisiteur() {
		return listeAttractionsVisiteur;   }
	@Override
	public String toString() {
		return "Visiteur [nomPrenom=" + nomPrenom +
				", age=" + age + ", taille=" + taille + "]";
	}
	public boolean addAttraction (Attraction attraction)
				throws AccesInterditException {
		if (!attraction.accesPossible(this))
			throw new AccesInterditException ("AccesInterdit : "
					+ this + " , "+ attraction);
		return listeAttractionsVisiteur.add(attraction);
	}
	@Override
	public int hashCode() { return Objects.hash(num);    }
	@Override
	public boolean equals(Object obj) {
		if (this == obj)  return true;
		if (obj == null)  return false;
		if (getClass() != obj.getClass())  return false;
		Visiteur other = (Visiteur) obj;
		return num == other.num;
	}
}
public class Attraction implements Comparable<Object>{
	private String nom ; private float cout ;
	private Integer ageMin ;  // peut etre null
	private Integer tailleMax ;  // peut etre null
	private HashSet<Visiteur> listVisiteursAttraction ;
	public Attraction(String nom, float cout, Integer ageMin,
				Integer tailleMax) {
		this.nom = nom;		this.cout = cout;
		this.ageMin = ageMin;	this.tailleMax = tailleMax;
		this.listVisiteursAttraction= new HashSet<Visiteur>();
	}
	public String getNom() {return nom;}
	public void setNom(String nom) {this.nom = nom;}
	public float getCout() {return cout;}
	public void setCout(float cout) {  this.cout = cout; }
	public Integer getAgeMin() {  return ageMin;}
```

```
	public void setAgeMin(Integer ageMin) {this.ageMin = ageMin;}
	public Integer getTailleMax() {return tailleMax;}
	public void setTailleMax(Integer tailleMax) {
			this.tailleMax = tailleMax;   }
	public HashSet<Visiteur> getListVisiteursAttraction() {
		return listVisiteursAttraction;}
	@Override
	public String toString() {
		return "Attraction [nom=" + nom + ", cout=" + cout +
		    ", ageMin=" + ageMin + ", tailleMax=" + tailleMax + "]";
	}
	public boolean accesPossible(Visiteur visiteur) {
		/*if (this.ageMin==null)
			if (visiteur.getTaille() < this.tailleMax) return true;
			else return false;
		if (this.tailleMax==null)
			if (visiteur.getAge() > this.ageMin) return true;
			else return false;
		if ( (visiteur.getTaille() < this.tailleMax)
			&& (visiteur.getAge() > this.ageMin) )
				return true;
		return false;*/
		return true;
	}
	public boolean addVisiteur
		(Visiteur visiteur) throws AccesInterditException {
		if (!this.accesPossible(visiteur))
			throw new AccesInterditException ("AccesInterdit : "
				+ this + " , "+ visiteur);
		return listVisiteursAttraction.add(visiteur);
	}
	@Override
	public int compareTo(Object o) {
		if ((o != null) &&
			(o.getClass().equals("Attraction") ) ) {
			Attraction other = (Attraction) o;
			return (int)(this.cout - other.cout);          }
		return 1;
	}
}
```

```
public class ParticipationAttraction  {
      private Visiteur visiteur;
      private Attraction attraction  ;
      private Date dateParticipation ;  // Date courante
      private static SimpleDateFormat formatter =
              new SimpleDateFormat ("dd/MM/yyyy", Locale.FRANCE);
      public ParticipationAttraction(Visiteur visiteur,
         Attraction attraction) throws AccesInterditException {
            this.visiteur = visiteur;this.attraction = attraction;
            this.dateParticipation= new Date();
            this.visiteur.addAttraction(this.attraction);
            this.attraction.addVisiteur(this.visiteur);
      }
      public ParticipationAttraction(Visiteur visiteur,
            Attraction attraction, Date dateParticipation)
                  throws AccesInterditException {
            this.visiteur = visiteur;this.attraction = attraction;
            this.dateParticipation= (Date) dateParticipation;
            this.visiteur.addAttraction(this.attraction);
            this.attraction.addVisiteur(this.visiteur);
      }
      public Visiteur getVisiteur() {return visiteur;}
      public void setVisiteur(Visiteur visiteur) {
            this.visiteur = visiteur;}
      public Attraction getAttraction() {return attraction;}
      public void setAttraction(Attraction attraction) {
            this.attraction = attraction; }
      public Date getDateParticipation() {return dateParticipation;}
      public void setDateParticipation(Date dateParticipation) {
            this.dateParticipation = dateParticipation;    }
      public String getDateParticipationString() {
            return (String)formatter.format (dateParticipation.getTime());}
      @Override
      public String toString() {
            return "ParticipationAttraction [visiteur=" +
                   visiteur.getNomPrenom() + ", attraction=" +
                  attraction.getNom() + ", dateParticipation="
                  + getDateParticipationString() + "]";
      }
}


public class ParticipationAttraction  {
```

```
	private Visiteur visiteur;
	private Attraction attraction  ;
	private Date dateParticipation ;  // Date courante
	private static SimpleDateFormat formatter =
			new SimpleDateFormat ("dd/MM/yyyy", Locale.FRANCE);
	public ParticipationAttraction(Visiteur visiteur,
				Attraction attraction) throws AccesInterditException{
		this.visiteur = visiteur;
		this.attraction = attraction;
		this.dateParticipation= new Date();
		this.visiteur.addAttraction(this.attraction);
		this.attraction.addVisiteur(this.visiteur);
	}
	public Visiteur getVisiteur() {return visiteur;}
	public void setVisiteur(Visiteur visiteur) {
				this.visiteur = visiteur;	}
	public Attraction getAttraction() {return attraction;}
	public void setAttraction(Attraction attraction) {
		this.attraction = attraction;}
	public Date getDateParticipation() {
		return dateParticipation;	}
	public void setDateParticipation(Date dateParticipation) {
		this.dateParticipation = dateParticipation;	}
	public String getDateParticipationString() {
		return (String)
				formatter.format (dateParticipation.getTime());}
	@Override
	public String toString() {
		return "ParticipationAttraction [visiteur=" + visiteur + ",
				attraction=" + attraction + ", dateParticipation="
					+ dateParticipation + "]";
	}
}
public class AccesInterditException extends Exception {
	public AccesInterditException(String msg) {
		super(msg);}
}
```

TP N°9 : Les fichiers en JAVA

Enoncé

Le but de ce TPest de développer une application de type gestion d'emploi du temps. Il s'agit ici d'afficher les séances de cours et de TP d'un groupe d'étudiants. Pour chaque séance de cours une séance de TP est prévue. Une séance est définie par le libellé de la matière, son type (CM ou TP), sa date, son heure de début et sa durée.

- L'application doit permettre de rajouter/supprimer/mettre à jour une séance de cours et la séance de TP correspondante.
- Les informations sont stockées sur le serveur dans un fichier « edt.txt »:
- Chaque ligne du fichier correspond à une séance : Elle contient les informations définissant la
- séance séparée par des « ; » selon la forme suivante :

  Libellé;dateCM;débutCM;duréeCM;dateTP;débutTP;duréeTP

1. Définir les classes permettant de modéliser une séance de Cours/TP.
2. Ecrire une classe qui modélise l'emploi du temps d'un groupe. Cette classe doit assurer les taches suivantes :
   - Ajout d'une séance de cours et de la séance de TP correspondante. Il faut ajouter la ligne adéquate au fichier « edt.txt»
   - Afficher la liste des séances dans leur ordre d'apparition dans le fichier.
   - Ordonnancement la liste des séances par Date ou par Matière (utiliser un paramètre ordre= « date » ou « matiere »).
   - Afficher la liste des séances pour un mois donné.
   - Afficher la liste des séances pour une matière donnée.
   - Afficher les séances semaine par semaine

Utilisation des fichiers Texte en JAVA

Java dispose d'une classe File qui offre des fonctionnalités de gestion de fichiers. La création d'un objet de type File permet de le faire associer à un fichier.

File nomfich = new File (nomFichier) ;

Si l'on dispose d'un objet de type File associé à un fichier ou à un répertoire, on peut tester son existence à l'aide de la méthode suivante : boolean exists ().

La classe FileWriter qui dérivée de la classe abstraite Writer (classe de base à toutes les classes relatives à un flux texte de sortie), permet de manipuler un flux texte associé à un fichier (Avec FileWriter , Si le fichier associé n'existe pas, il est créé (vide). S'il existe, son ancien contenu est détruit).

Si l'on souhaite disposer de possibilités de formatage, on peut recourir à la classe PrintWriter qui dispose d'un constructeur recevant en argument un objet de type FileWriter: La classe PrintWriter est dotée, entre autres, des méthodes print et println, analogues à celles de la classe System.out. Elles permettent d'enregistrer dans un fichier texte des informations (formatées) d'un type primitif quelconque.

```
PrintWriter sortie = new PrintWriter (new FileWriter (nomfich)) ;
sortie.println (…) ;
sortie.close () ;
```

La classe FileReader en la couplant avec la classe BufferedReader qui dispose d'une méthode readLine, permet lire chacune des lignes du fichier texte.

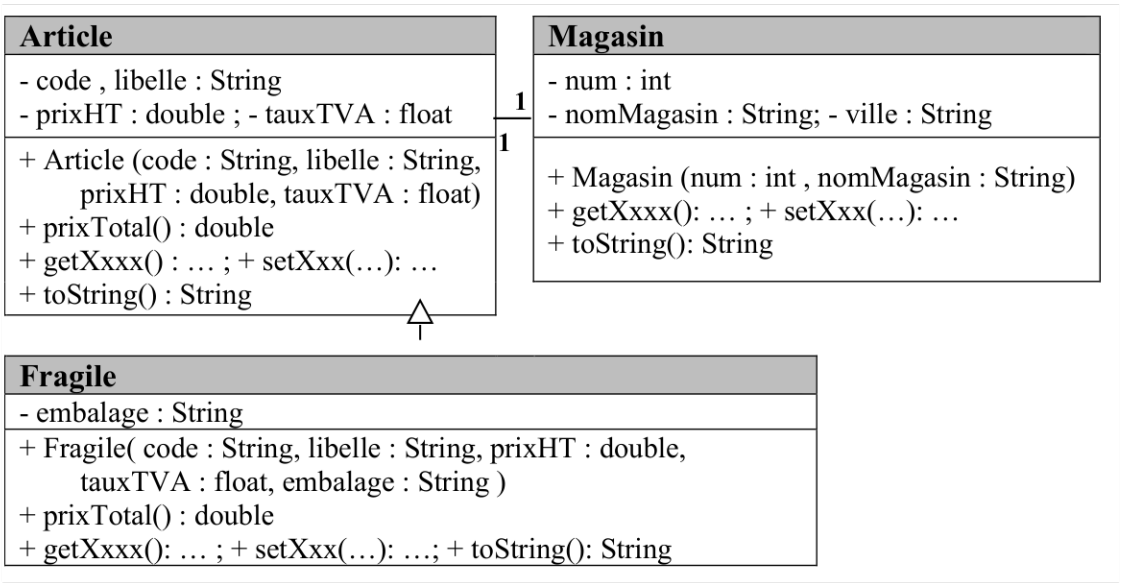
```
BufferedReader entree = new BufferedReader (new FileReader (nomfich)) ;
Do {
        ligne = entree.readLine() ;
        if (ligne == null) break ;
        …
}while (ligne != null) ;
entree.close () ;
```

Java il dispose d'une classe utilitaire nommée StringTokenizer (package java.util), qui permet de découper une chaîne en différents tokens (sous-chaînes), en se fondant sur la présence de caractères séparateurs qu'on choisit librement.

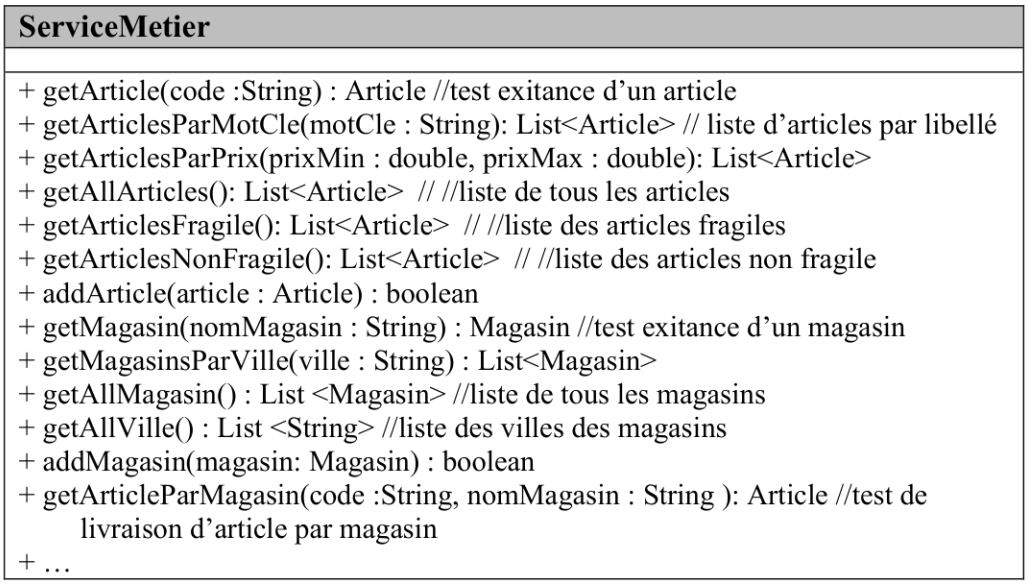
```
StringTokenizer tok = new StringTokenizer (ligne, ";") ;
// Le 2 argument indique les séparateurs
tok.nextToken() ; // fournit le token suivant s'il existe
```

TP N°10 : Collections et bases de données

Dans cet examen nous nous intéressons à une application web JAVA de gestion de vente d'articles électroménagers de magasins placées dans différentes villes du Royaum. La figure suivante représente un extrait du diagramme de classes élaboré lors de la phase d'analyse/conception de cette application.



Les différents services metiers associés à notre application sont regroupés dans une classe « ServiceMetier ». La structure de cette classe est donnée par le diagramme suivant (toutes les méthodes sont considérées static) :



A. Considérons, dans un premier lieu, les classes POJO « Articles » , « Magasin » et « Fragile ».

1. Ecrire le code du constructeur de la classe « Fragile »
2. Redéfinir la méthode « prixTotal() » de la classe « Fragile » sachant que le prix total normal d'un article est « prixTotal=prixHT*(1+tauxTVA) » alors que celui des articles fragiles est « prixTotal=prixHT*(1+tauxTVA) * 1.1 »

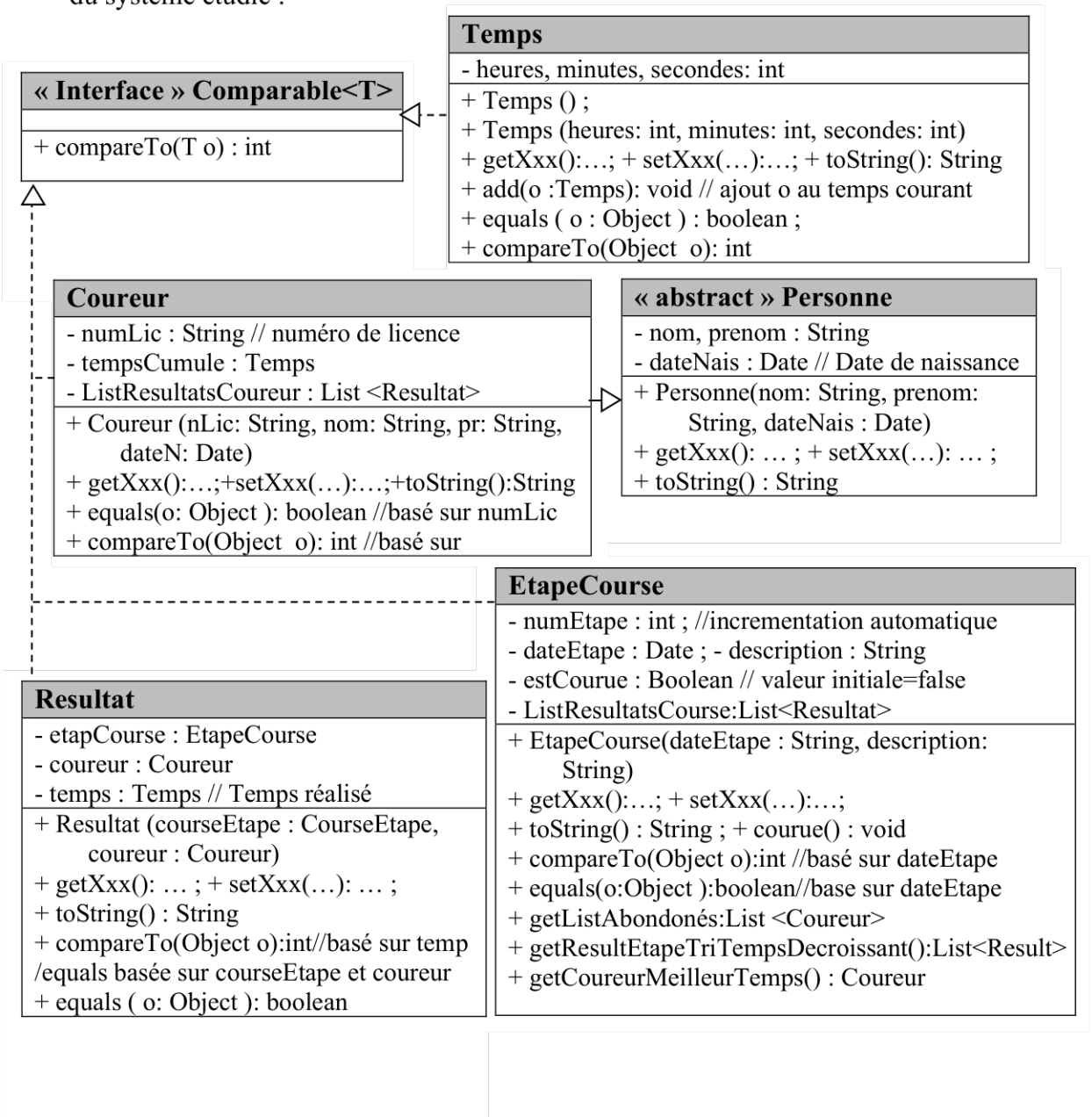
B. Considérons que les informations relatives aux articles et aux magasins sont stockées dans une base de données MYSQL nommée « BDArticles ». Les diférents services DAO , utilisant JDBC, sont regroupés dans une classe « ServiceDAO ». La structure de cette dernière est similaire à celle de la classe « ServiceMetier » (on suppose qu'méthode de la classe « ServiceDAO » est appelée par la méthode de la classe « ServiceMetier » lui correspond et ayant la même signature).

1. Définir une classe « Connexion » permettant d'obtenir une connexion unique (instance unique pour l'ensemble des connexions) à la base de données « BDArticles »
2. Ecrire le code de la méthode DAO « addMagasin » permattant l'ajout d'un nouvel enregistrement dans la table « magasin »
3. Donner le code de la méthode DAO « getMagasinsParVille» permattant de retourner la liste des magasins d'une ville donnée

TP N°11 : Accès aux données et services métiers

On s'intéresse dans ce contrôle à une application JAVA de gestion des classements du Tour du Maroc. Ce Tour est une course à étapes dont le résultat d'un coureur est déterminé à la fin de chaque étape par le temps réalisé lors de de l'étape et le cumul des précédents temps réalisés. Pour des raisons de simplification on se limite classements associés au meilleur temps réalisé lors d'une étape (maillot vert) et au meilleur temps cumulé ("maillot jaune").

A. Considérons dans un premier lieu un diagramme modélisant certaines classes métiers du système étudié :



- La classe « Temps » modélise le temps réalisé par un coureur. La méthode toString de cette classe retourne l'heure sous la forme "xhymnzsec".
- L'instanciation de la classe « Coureur » permet de représenter un coureur qui participe au courses (le numéro de licence est supposée unique pour chaque coureur).
- La classe « EtapeCourse » simule le déroulement d'une étape de course (deux étapes ne peuvent pas avoir lieu le même jour). La méthode courue de cette classe permet de considérer l'étape comme courue (affectation de la valeur true à l'attribut estCourue).
- La méthode «getListAbondonés» return la liste des coureurs ayant abandonné au cours de l'étape. Un coureur est considéré abandonné si le temps de son résultat est null (l'attribut temps n'a pas été modifié après instanciation).
- Les méthodes «getListAbondonés», «getResultEtapeTriTempsDecroissant» et «getCoureurMeilleurTemps» lève l'exception «EtapeNonCourueException» si l'étape n'a pas encore été courue.
- La participation d'un coureur dans une étape de course est donnée par une instance de la classe « Resultat ». Dans ce cas l'attribut temps est initialisé par la valeur null. Lorsque le résultat du coureur est connu, le temps réalisé est affecté à l'attribut temps (méthode SetTemps).

B. Les informations relatives aux coureurs, celle des étapes de course et les résultats sont stockées dans une base de données MYSQL nommée « BDTour »). Les différents services DAO (d'accés à la base de données) utilisant JDBC, sont regroupés dans une classe « ServiceDAO ». On considère une classe « Connexion » permettant d'obtenir une connexion unique (instance unique pour l'ensemble des connexions) à la base de données « BDTour» . Pour faciliter l'accès à ces informations, elles ont été regroupées dans une classe «**ServicesMetier**».

ServicesMetier
- listEtapeCourse : List<EtapCourse> - listCoureur : List<Coureur> - listResultas : List <Resultat> - estTermine : Boolean // valeur initiale=false
+ Services() ; + getXxx(): … ; + termine() : void // tour terminé : estTermine = true + getCoureurByLicence(numLic :String) : Coureur + getEtapeCourseByDate(dtCourse:Date):EtapCourse + getListEtapeCourseCourue():List<EtapCourse> + getListEtapeCourseNonCourue():List<EtapCourse> + getResult(etap:EtapeCourse,coureur:Coureur):Result + getCoureursMeilleurTempsCourse():List<Coureur> + getCoureurMeilleurTempsCumuleEtapes():Coureur + getCoureurMeilleurTempsCumuleTour():Coureur + addCoureur(coureur : Coureur) : void + addEtapCourse (etapCourse : EtapCourse) : void + addParticipation(etape:CourseEtape, coureur: Coureur):void + addResultat (courseEtape : CourseEtape, coureur: Coureur, temps : Temps) : void …

ServicesMetier
+ getAllCoureur():List<Coureur> + getAllEtapeCourse(): List<EtapCourse > + getAllResult():List<Result> + addCoureur(coureur:Coureur):void + addEtapCourse(etapCourse: EtapCourse) : void + addParticipation (resultat: Resultat):void + addResultat(resultat :Resultat):void …

- La méthode « addParticipation » permet l'ajout d'une nouvelle instance de la classe « Resultat » à la liste « listResultas ». Le temps associé à l'instance ainsi crée (attribution de résultat) sera définie ultérieurement par la méthode « addResultat »

- « getCoureurMeilleurTempsCumuleTour» lève l'exception « TourNonTermineException» si le Tour n'est pas encore terminé.
- Les méthodes « addCoureur », « addEtapCourse », « addParticipation » et « addResultat » lèvent l'exception « TourTermineException » si le Tour est terminé.

Schéma de la base de donnée

- personnes(numLic, nom, prenom, dateNais, hTempsCumul, mnTempsCumul , sTempsCumul, typePersonne)
- etapeCourses (numEtape, dateEtape, description, estCourue)
- resultat(numRes, numEtape#, numLic#, hTemps, mnTemps , sTemps)
- resultTour(num, numEtape, CoureurMeilleurTemps, meilleurTemps, termine) //termine = true si le tour est terminé