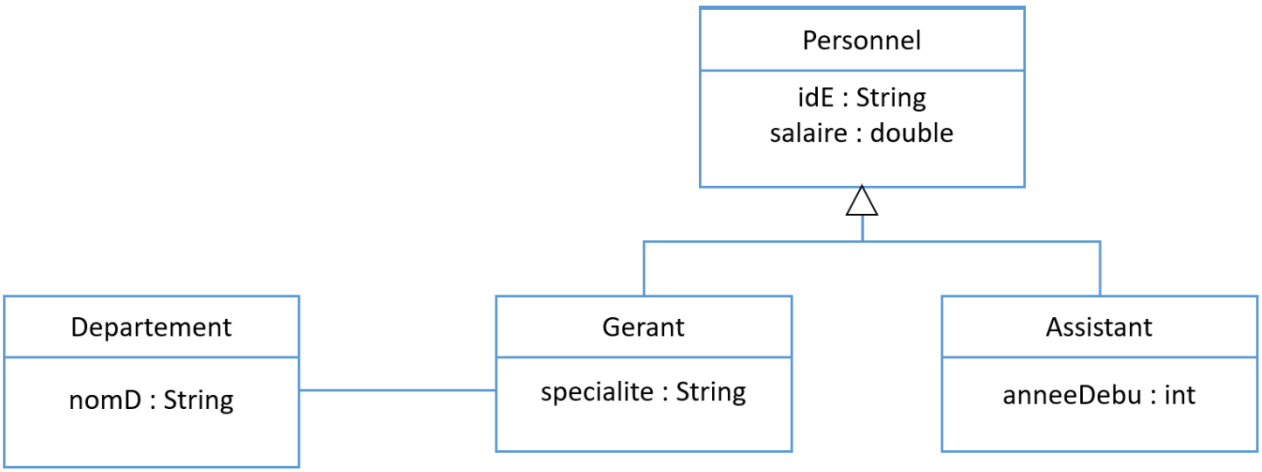
Révision : Développement Java avancé

Réalisez ces exercices sans l'aide de l'ordinateur pour évaluer vos compétences d'analyse, puis, implémentez vos codes pour tester si votre réponse est correcte

Exercice 1 partie 1 :

1- Implémentez en Java les classes suivantes tenant en considération les relations entre les classes « ***one-to-many*** » ou bien « ***one-to-one*** », pensez à créer les constructeurs nécessaires sachant que le constructeur de la classe mère contient tous ses attributs.
2- Il est nécessaire de décrire le type de relation dans le code, c.-à-d. il faut déduire le type de la relation juste en lisant le code.

NB : un gérant peut gérer plusieurs départements, un département ne peut être géré que par un seul gérant.



3- Créez 2 objets de chacune des classes (sauf la classe mère).

Exercice 1 partie 2 (suite de la partie précèdente) :

Considérons les classes du diagramme précèdent. Soit les collections suivantes.
List < Personnel > **lesPrs** ;

Set < Departement > **lesDeps**;

La liste **lesPrs** contient des **Gérants** ainsi que des **Assistants**.
Ces collections sont déjà remplies par des objets.

1- Ecrivez une méthode qui prend les paramètres nécessaires puis ajoute un département au set **lesDeps**.
NB : chaque département doit voir un nom unique. Et, pour ajouter un département il faut l'affecter à un gérant.

2- Ecrivez une méthode statique qui permet de supprimer un département du set.
NB : n'oublier pas de traiter la relation entre département et gérant lors de la suppression.

3- On désire regrouper le nombre d'assistants par année de début. Ecrivez le instructions permettant de créer une **Map < … , … >** associant à chaque salaire le nombre d'assistants correspondant. (**salaire** : nombre d'assistants ayant le même salaire)

4- On désire regrouper les **Gerants** stockés dans la liste '**lesPrs**' ayant un **salaire inférieur à 10000** par **spécialité**. Ecrivez une méthode qui prend les paramètres nécessaires puis retourne une Map < … , … > associant chaque **spécialité** aux **gérants** correspondants.

Exercice 3 : Java Streams

Soit la classe **Produit** définit par les attributs suivants : (id : String, nom : String, prix : double, categorie : String, anneFabric : int).

- L'id d'un produit est un construit de chiffres.
- Les catégories existantes sont : Electronique, meuble, librairie

Supposant qu'on a un Set<Produit> contenant plusieurs objets de la classe Produit.

Créez **un seul** Stream permettant d'effectuer l'objectif suivante :

Appliquer une réduction de 15% sur les produits dont le prix est inferieure à 2000 de la catégorie électronique. Dans la suite, on ne doit pas changer la forme des id des produits de la catégorie **Librairie** et **Jardin**, pour les autres catégories, on veut changer la forme des id de manière à ce que l'id de chaque produit contiendra les trois premières lettres de la catégorie suivi de son numéro id séparé par un tiré. (**Exemple** : pour un produit dont l'id est **12** de la catégorie **Electronique**, son id devient : **ELE-12**.). Les produis traités doivent être stockés dans une collection accessible par des indices.

Exercice 4 : Exceptions

En Java, `NullPointerException` est une exception qui peut subvenir lors de l'un des cas suivants :

- Mauvaise utilisation d'une référence à un objet.
- Accès d'un objet déclaré mais avec la valeur **null**.
- Acces au elements d'un tableau d'objets crées mais pas encore initialisés.

Question : vous devez comprendre le code suivant, puis, ajoutez les instructions nécessaires pour que le code affiche le message « Erreur Null pointer trouvée » lorsqu'il catche une `NullPointerException.`

```
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
Etudiant best = students[0];
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...

for(Etudiant e : students) {
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
     if(e.moy > best.moy) {
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
          best = e;
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
     }
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
}
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
System.out.println(" "+best.moy);
.. .. .. .. .. .. .. .. .. .. .. .. .. .. ...
```

Exercice 5 : ThreadPoolExecutor / ExecutorService

Pour réviser les threads il faut répondre aux questions suivantes.

1- Quelle sont les critères à prendre en considération avant d'adopter une solution basée sur une architecture **ThreadPoolExecutor/ExecutorService ?**
2- Comment déclarer un **ThreadPoolExecutor/ExecutorService** permettant d'exécuter 7 threads en parallèle.
3- Que représente une classe qui héritent de la classe **Thread** dans une architecture **ThreadPoolExecutor/ExecutorService?**
4- Quel est l'utilité de la méthode **run()** ?
5- Expliquez l'utilité de la méthode **submit()** dans le cadre des **ThreadPoolExecutor/ExecutorService** ?
6- Qu'est-ce qu'une file d'attente ?
7- Quels sont les problemes d'execution qui peuvent causés par l'utilisation d'une architecture **ThreadPoolExecutor/ExecutorService ?**
8- Expliquez l'utilisation de : **Collection< Future <?> >**.

Exercice 6 : multi-threading Synchronization

Pour réviser la synchronisation, Il faut bien comprendre l'exercice 1 du TP5 (**Synchronisation**), puis, répondre aux questions suivantes :

1- Expliquez l'importance de la synchronisation dans un environnement multithreading.
2- Expliquez le problème que la synchronisation des threads peut résoudre. C.-à-d., dans quels cas on doit utiliser la synchronisation.
3- Quel est l'utilité de **Lock** et comment l'utiliser **?**
4- Quel est l'utilité de « **synchronized** » ?
5- Comment s'assurer que votre code cause un problème de manque de synchronisation ?
6- Quels sont les points clé à prendre en considération pour mettre en place une solution de synchronisant l'exécution concurrente.
7- Qu'est-ce que la section critique d'un code ?
8- Qu'est-ce que « **start()** » et « **join()** » ?

Exercice 7 : Threads

Le code ci-dessous permet d'exécuter des requêtes SQL par des threads dans une base de donnée MySQL.
Votre objectif est de comprendre le code, puis, répondre aux questions suivantes

```
public class TaskExeQuery extends Thread{

	public Connection connection;
	public String query;
	public TaskExeQuery(Connection connection, String query) {
		this.connection = connection;
		this.query = query;
	}

	public void run() {
		Operations.execQuery(query, connection);
	}
}
```

```
public class Operations {

  public static void execQuery(String query, Connection con) {

		Statement statement=null;
		ResultSet resultSet=null;
		try {
			statement = con.createStatement();
			resultSet = statement.executeQuery(query);
			while (resultSet.next()) {
				String valCol = resultSet.getString("hometown");
				System.out.println(valCol);
			}
		} catch (SQLException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
  }
}
```

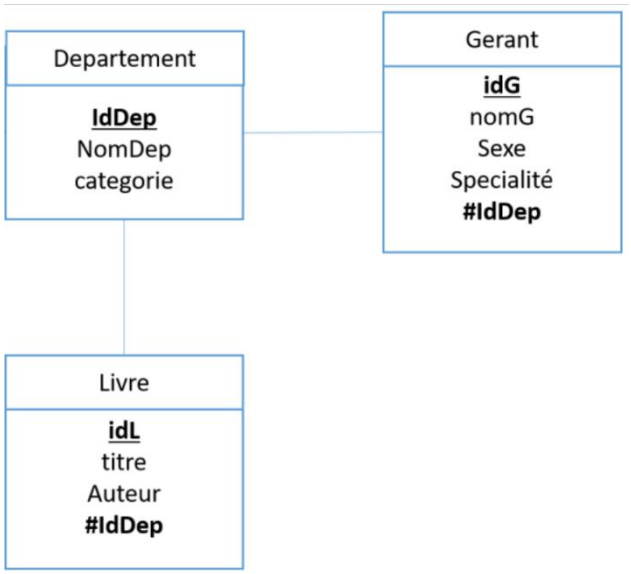
1- Modifiez la déclaration de la méthode « *execQuery* » afin de garantir qu'elle ne puisse être exécutée que par un seul thread à la fois (c'est-à-dire la rendre sécurisée en contexte concurrent).
   a. En utilisant **Synchronized**
   b. En utilisant **Lock**
2- Dans la classe « **MainEx** » écrivez les institutions permettant l'exécution des deux threads **th1** et **th2**, le thread **th2** doit attendre la fin d'exécution du thread **th1.**

```
public class MainEx {
 public static void main(String[] args) {
  String url = "jdbc:mysql://127.0.0.1/ourDataBase";
  String user = "root";
  String password = "";
  Connection connection=null;
  try {
       connection = DriverManager.getConnection(url, user, password);
  } catch (SQLException e) {
       e.printStackTrace();
  }
  TaskExeQuery task1 = new TaskExecuteQuery(connection, "SELECT * FROM membre");
  TaskExeQuery task2 = new TaskExecuteQuery(connection, "SELECT hm FROM membre");

  Thread th1 = new Thread(task1);
  Thread th2 = new Thread(task2);

  // Reponse à la question 2
```

3- Expliquez l'utilité de :
   a. `DriverManager`
   b. `jdbc:mysql://`

Exercice 7 : Hibernate

Soit le schéma relationnel suivant :



Observez les classes suivantes et répondez aux questions (voir la page suivante).

```
import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "departement")
public class Departement {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "idDep")
    private Long idDep;

    @Column(name = "nomDep", nullable = false)
    private String nomDep;

    @Column(name = "categorie")
    private String categorie;

    @OneToMany(mappedBy = "departement", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<Livre> livres;

    public Departement() {}

    // Getters & Setters
}
```

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import java.util.List;

public class DepartementDAO {

    private SessionFactory sessionFactory;

    public DepartementDAO(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    public void save(Departement departement) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();
        session.save(departement);
        tx.commit();
        session.close();
    }

    public Departement findById(Long id) {
        Session session = sessionFactory.openSession();
        Departement dep = session.get(Departement.class, id);
        session.close();
        return dep;
    }

    public List<Departement> findAll() {
        Session session = sessionFactory.openSession();
        List<Departement> list = session.createQuery("from Departement",
Departement.class).list();
        session.close();
        return list;
    }

    public void delete(Departement departement) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();
        session.delete(departement);
        tx.commit();
        session.close();
    }
}
```

1- Expliquez l'intérêt de chacune des Annotations de la classe Département.
2- Selon la classe Département, donnez la déclaration complète de la classe Livre.
3- Quel est le rôle de : `cascade = CascadeType.ALL`
4- Dans la classe DepartementDAO, expliquez le role de : *Transaction*, *beginTransaction*, *createQuery, commit, SessionFactory.*
5- Dans la classe DepartementDAO, ajoutez la méthode *updateDepartement(…)* en utilisant les méthodes Hibernate **standards.**
6- **Expliquez l'utilité des concepts et outils suivants en développement java :** Jakarta, persistance des données, ORM.