TD4 - Packages

GAUTIER Titouan

1 - Etablir la connexion

La première etape est d'établir la connexion avec notre base de données.

```
package database

import java.sql.Connection
import java.sql.DriverManager
import java.sql.SQLException
import java.util.*

class SessionOracle {

    var conn: Connection? = null
    var username: String = "i2a01a" // provide the username
    var password = "titouangautier" // provide the corresponding password
    var database="pdb1"  // le nom de la base


    /**
     * This method makes a connection to Oracle  Server
     */


    fun getConnectionOracle(): Connection? {
        val connectionProps = Properties()
        connectionProps["user"] = username
        connectionProps["password"] = password

        try {
            Class.forName("oracle.jdbc.driver.OracleDriver").newInstance()
            conn = DriverManager.getConnection("jdbc:oracle:thin:@172.26.82.68:1521:pdb1",username, password);
            println("connexion réussie")
        } catch (ex: SQLException) {
            // handle any errors
            ex.printStackTrace()
        } catch (ex: Exception) {
            // handle any errors
            ex.printStackTrace()
        }
        return conn
    }


}
```

Pour cela, nous réalisons une classe SessionOracle qui comporte 4 attribut :

- L'attribut conn de type Connexion dans lequel nous définirons la connexion mais qui pour l'instant est nul.
- L'attribut username qui est le pseudo utiliser pour se connecter à la BD.
- L'attribut password qui est le password utilise pour se connecter à la DB.
- L'attribut database qui est le nom de la BD.

Nous définissons une méthode getConnectionOracle qui définira la connexion à la BD. Cette méthode récuper le username et le password et il renvoie la connexion ou une exception si il y a une erreur.

On peut tester notre fonction en créant une instance de SessionOracle dans la fonction Main et en executant la méthode getConnectionOracle. Nous reproduirons cette démarche pour tester à chaque nouvelle méthode.

2 - La classe Employe

Maintenant, il faut créer une classe employe pour pouvoir donner un type au employé récupérer dansla base de données.

```
class Employe(nuempl: Int, nomempl: String, hebdo: Int, affect: Int, salaire : Int) {
    private var nuempl : Int
    private var nomempl : String
    private var hebdo: Int
    private var affect: Int
    private var salaire: Int

    init {
        this.nuempl = nuempl
        this.nomempl = nomempl
        this.hebdo = hebdo
        this.affect = affect
        this.salaire = salaire
    }

    fun getNuempl() : Int  {
        return this.nuempl
    }


    fun getNomempl() : String {
        return this.nomempl
    }

    fun getHebdo() : Int  {
        return this.hebdo
    }

    fun getAffect() : Int  {
        return this.nuempl
    }

    fun getSalaire() : Int {
        return this.salaire
    }

    fun setNomempl(nom : String) {
        this.nomempl = nom
    }

    fun setHebdo(h : Int) {
        this.hebdo = h
    }

    fun setAffect(a : Int) {
        this.hebdo = a
    }

    fun setSalaire(s : Int) {
        this.salaire = s
    }

    override fun toString() : String {
        return "$nuempl $nomempl $hebdo $affect"
    }

}
```

Dans cette classe nous définissons les getters et les setters. C’est à dire des méthodes qui nous permettrons de récupérer et redéfinir les attributs de la classe Employé. Nous redéfinissons aussi la méthode toString qui nous permettera de print nos employé plus facilement.

3 - Récupérer les donnés

3.1 - Le package DAO

Ensuite, il faut qu’on récupère les données dans notre BD. Pour cela nous allons créer un nouveau package DAO avec une nouvelle classe DAOEmployer, cette classe prend en paramètre une instance de SessionOrcale. Cette classe comportera 4 méthode:

- create, qui nous permettera d’insérer un employe dans la BD
- delete, qui permet de supprimer un employe dans la BD
- update, qui permet de modifier un employe dans la BD

Ces 3 méthodes prennent en paramètre un employé. Il reste une dernière méthode read qui nous permettera d'afficher toutes notre table employé de la BD.

```
package DAO
import Bean.Employe
import database.SessionOracle
import java.sql.Connection
import java.sql.*

class DAOEmploye(val ss: SessionOracle) {
    var session: SessionOracle? = null
    init {
        this.session=ss
    }

    fun create(e : Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete: String = "Insert into employe VALUES(${e.getNuempl()},'${e.getNomempl()}',${e.getHebdo()},${e.getAffect()},${e.ge

        try {
            val stmt: Statement = conn!!.createStatement()// Création d'une requete de type Statemen
            stmt.executeUpdate(requete)
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }

    fun update(e : Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete : String = "UPDATE employe SET nomempl = '${e.getNomempl()}', hebdo = ${e.getHebdo()}, affect = ${e.getAffect()},

        try {
            val stmt: Statement = conn!!.createStatement()// Création d'une requete de type Statemen
            stmt.executeUpdate(requete)
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }

    fun delete(e: Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete : String = "DELETE from employe where nuempl = ${e.getNuempl()}"

        try {
            val stmt: Statement = conn!!.createStatement()// Création d'une requete de type Statemen
            stmt.executeUpdate(requete)
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }


    fun read(){


        //var essai = SessionOracle();
        var conn: Connection? = null
        conn= session?.getConnectionOracle()
        val requete: String="SELECT * FROM employe"
        try {
            val stmt: Statement = conn!!.createStatement()// Création d'une requete de type Statemen
            val result: ResultSet= stmt.executeQuery(requete) //Le contenu du select est dans ResultSet

            /* Parcourir le résultat du select avec la fonction next();*/
            while (result!!.next()) {

                // getting the value of the id column
                val id = result.getInt("nuempl")
                val nom=result.getString("nomempl")
                val hebdo = result.getInt("hebdo")
                val affect = result.getInt("affect")
                val salaire = result.getInt("salaire")
                println("$id $nom $hebdo $affect $salaire")

            }
            result.close()
```

```
        }

        catch(e: SQLException){
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
    }
}
```

Dans chaque méthode nous allons définir la requête SQL dans une variable de type String. Ensuite, nous allons utilisé des variables de types Statement et des méhodes createStatement et executeUpdate ou executeQuery. La méthode createStatement nous permet de créer une requête SQL de type Statement. La méthodes executeUpdate prend en paramètre notre requête string et execute la requête dans la BD, elle est utilisé pour les insert, update et delete. La méthode executeQuery fait la même chose mais elle est utilisé pour des select.

3.2 - Le classe DAOEmployeBis

Cette nouvelle classe est la même que la précédente, cependant nous avons remplacer les Statement par des PreparedStatement et les createStatement par des prepareStatement.

```
package DAO
import Bean.Employe
import database.SessionOracle
import java.sql.Connection
import java.sql.*

class DAOEmployebis(val ss: SessionOracle) {
    var session: SessionOracle? = null
    init {
        this.session=ss
    }

    fun create(e : Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete: String = "Insert into employe VALUES(?,?,?,?,?)"

        try {
            val stmt: PreparedStatement = conn!!.prepareStatement(requete)// Création d'une requete de type Statemen
            stmt.setInt(1,e.getNuempl())
            stmt.setString(2,e.getNomempl())
            stmt.setInt(3,e.getHebdo())
            stmt.setInt(4,e.getAffect())
            stmt.setInt(5,e.getSalaire())
            stmt.executeUpdate()
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }

    fun update(e : Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete : String = "UPDATE employe SET nomempl = ?, hebdo = ?, affect = ?, salaire = ? where nuempl = ? "

        try {
            val stmt: PreparedStatement = conn!!.prepareStatement(requete)// Création d'une requete de type Statemen
            stmt.setString(1,e.getNomempl())
            stmt.setInt(2,e.getHebdo())
            stmt.setInt(3,e.getAffect())
            stmt.setInt(4,e.getSalaire())
            stmt.setInt(5,e.getNuempl())
            stmt.executeUpdate()
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }

    fun delete(e: Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete : String = "DELETE from employe where nuempl=?"

        try {
            val stmt: PreparedStatement = conn!!.prepareStatement(requete)// Création d'une requete de type Statemen
```

```
            stmt.setInt(1,e.getNuempl())
            stmt.executeUpdate()
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }


    fun read(){


        //var essai = SessionOracle();
        var conn: Connection? = null
        conn= session?.getConnectionOracle()
        val requete: String="SELECT * FROM employe"
        try {
            val stmt: Statement = conn!!.createStatement()// Création d'une requete de type Statemen
            val result: ResultSet= stmt.executeQuery(requete) //Le contenu du select est dans ResultSet

            /* Parcourir le résultat du select avec la fonction next();*/
            while (result!!.next()) {

                // getting the value of the id column
                val id = result.getInt("nuempl")
                val nom=result.getString("nomempl")
                val hebdo = result.getInt("hebdo")
                val affect = result.getInt("affect")
                val salaire = result.getInt("salaire")
                println("$id $nom $hebdo $affect $salaire")

            }
            result.close()
        }

        catch(e: SQLException){
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
    }
}
```

Ce qui change c'est que les variables présente dans notre requête SQL vont être remplacé par des point d'interrogation. Nous allons remplacé ces point d'interrogation, apres avoir créer la requête PreparedStatment, grace à la méthode setInt et setString. Ces méthodes prennent 2 paramètres: le premier est la position du point d'interrogation que l'on veut remplacer, le deuxième est la valeur par laquelle on veut le remplacer. Cette manière de procéder va permettre d'empécher les injections SQL et donc sécurisé notre système.

3.3 - La classe DAOEmployeTer

Cette classe reprend le fonctionnement de la dernière, mais cette fois-ci, au lieu de faire des requête SQL simple, nous allons faire appels à des procédures stockées dans la BD, ce sont des méthode créer au préalable qui permettent de faire certaines requête à la BD. Nous allons les appeler grace à la méthode call. Les PreparedStatement sont remplacé par des CallableStatement et les prepareStatement par des prepareCall.

```
package DAObis

import Bean.Employe
import database.SessionOracle
import oracle.jdbc.OracleTypes
import java.sql.*

class DAOEmployeter(val ss: SessionOracle) {
    var session: SessionOracle? = null
    init {
        this.session=ss
    }

    fun create(e : Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete: String = "call MAJ.CREER_EMPLOYE(?,?,?,?,?)"
        try {
            val stmt: CallableStatement = conn!!.prepareCall(requete)// Création d'une requete de type Statemen
            stmt.setInt(1,e.getNuempl());
            stmt.setString(2,e.getNomempl());
            stmt.setInt(3,e.getHebdo());
```

```
            stmt.setInt(4,e.getAffect());
            stmt.setInt(5,e.getSalaire());
            stmt.execute()
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }

    fun update(e : Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete : String = "UPDATE employe SET nomempl = ?, hebdo = ?, affect = ?, salaire = ? where nuempl = ? "

        try {
            val stmt: PreparedStatement = conn!!.prepareStatement(requete)// Création d'une requete de type Statemen
            stmt.setString(1,e.getNomempl())
            stmt.setInt(2,e.getHebdo())
            stmt.setInt(3,e.getAffect())
            stmt.setInt(4,e.getSalaire())
            stmt.setInt(5,e.getNuempl())
            stmt.executeUpdate()
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }

    fun delete(e: Employe) {
        var conn: Connection? = null
        conn = session?.getConnectionOracle()
        val requete : String = "DELETE from employe where nuempl=?"

        try {
            val stmt: PreparedStatement = conn!!.prepareStatement(requete)// Création d'une requete de type Statemen
            stmt.setInt(1,e.getNuempl())
            stmt.executeUpdate()
        } catch (e: SQLException) {
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
        this.read()
    }


    fun read(){


        //var essai = SessionOracle();
        var conn: Connection? = null
        conn= session?.getConnectionOracle()
        val requete: String="call LECTURE.liste_employes(?)"
        try {
            val stmt: CallableStatement = conn!!.prepareCall(requete)// Création d'une requete de type Stateme
            stmt.registerOutParameter(1,OracleTypes.CURSOR)
            stmt.execute()
            var result = stmt.getObject(1) as ResultSet

            /* Parcourir le résultat du select avec la fonction next();*/
            while (result!!.next()) {

                // getting the value of the id column
                val id = result.getInt("nuempl")
                val nom=result.getString("nomempl")
                val hebdo = result.getInt("hebdo")
                val affect = result.getInt("affect")
                val salaire = result.getInt("salaire")
                println("$id $nom $hebdo $affect $salaire")

            }
            result.close()
        }

        catch(e: SQLException){
            println(e.errorCode)//numéro d'erreur
            println(e.message)// message d'erreur qui provient d'oracle, trigger ou procédure
        }
    }


}
```