AIR-AFPA

Activité-type 1 : Développer une application client/serveur

Situation 4 : Coder en SQL

Lambert Benjamin
Mars 2017

Introduction

AIR-AFPA est une filiale d'AFPA TRAVEL France créée en décembre 2012.Elle s'est spécialisée dans le marché low-cost du transport de passagers. C'est dans le but de se faire une place sur ce marché qu'AIR-AFPA à été crée car d'ici 2020 le nombre de vol low-cost devrait s'accroître de 50 %.

Jusqu'en mi-2013, la filiale avait recours aux logiciels de sa maison-mère, mais ses besoins changeants elle souhaite développer son propre système d'information, c'est dans cette optique que le nouveau site (air-afpa.fr) a été inauguré en millieu d'année, et que le service des vols demande à présent la réalisation d'une application de gestion des vols.

1-Création du Singleton

Le singleton permet de s'assurer qu'une seule instance d'un objet donné sera instanciée pendant toute la durée de votre application.

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;


//la classe Connection BDD est le Singleton
public class ConnectionBDD {

    // ici toutes les variables necessaires à la connection
    private static final String DB_URL = "jdbc:mysql://localhost:3306/airafpa";
    private static final String DB_JDBC_DRIVER = "com.mysql.jdbc.Driver";
    private static final String DB_USER = "ProjetEval14";
    private static final String DB_PASSWORD = "projeteval14";

    //on initialise  un objet de type Connection à null
    private Connection cn = null;

    private ConnectionBDD() {

        try {

            Class.forName(ConnectionBDD.DB_JDBC_DRIVER);

        } catch (ClassNotFoundException ex) {

            ex.printStackTrace();
            System.exit(1);

        }

    }

    public static ConnectionBDD getInstance() {
        return ConnectionBDDHolder.INSTANCE;
    }

    private static class ConnectionBDDHolder {

        private static final ConnectionBDD INSTANCE = new ConnectionBDD();
    }
```

```java
public boolean connect() {
    /*
    si on a jamais été connecté on se connecte,
    sinon refresh la connection
    */

    if (this.cn == null) {

        try {

            this.cn = DriverManager.getConnection(ConnectionBDD.DB_URL, ConnectionBDD.DB_USER, ConnectionBDD.DB_PASSWORD);

        } catch (SQLException ex) {
            ex.printStackTrace();
            return false;

        }
    } else {

        try {

            Statement st = this.cn.createStatement();
            String requete = "SELECT 1";
            st.executeQuery(requete);
            //si ici ca ne marche pas c'est quon est  plus connecté, alors on se reconnecte
        } catch (SQLException ex) {
            try {

                this.cn = DriverManager.getConnection(ConnectionBDD.DB_URL, ConnectionBDD.DB_USER, ConnectionBDD.DB_PASSWORD);

            } catch (SQLException ex1) {

                ex1.printStackTrace();
                return false;

            }

        }

    }

    return true;
}
    public Connection getConnectionManager() {

        return this.cn;

    }
}
```

2-Mise en place du DAO

```java
import bddsql.ConnectionBDD;
import java.util.ArrayList;

public abstract class DAO<T,S> {

    protected ConnectionBDD bddmanager = null;

    public DAO() {

        this.bddmanager = ConnectionBDD.getInstance();
    }

    //equivalent à un insernt into T
    public abstract T creer(T obj);

    //equivalent à un delete from T
    public abstract boolean supprimer(S id);

    //equivalent à un select * from T
    public abstract ArrayList<T> getAll();

    //equivalent à u select * from T WHERE
    public abstract T find(S id);

    // equivalent à un Update
    public abstract T update(S id ,T obj);
}
```

3-Création de la classe Airport

```java
import java.util.Objects;

/**...4 lines */
public class Airport {

    private String code_AITA;
    private String city;
    private String country;

    public Airport(String code_AITA, String city, String country) {

        this.code_AITA = code_AITA;
        this.city = city;
        this.country = country;

    }

    public Airport() {

    }

    public Airport(String code_AITA){

        this.code_AITA=code_AITA;

    }

    public String getCode_AITA() {
        return code_AITA;
    }

    public void setCode_AITA(String code_AITA) {
        this.code_AITA = code_AITA;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }
```

```java
    @Override
    public int hashCode() {
        int hash = 5;
        hash = 41 * hash + Objects.hashCode(this.code_AITA);
        hash = 41 * hash + Objects.hashCode(this.city);
        hash = 41 * hash + Objects.hashCode(this.country);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Airport other = (Airport) obj;
        if (!Objects.equals(this.code_AITA, other.code_AITA)) {
            return false;
        }
        if (!Objects.equals(this.city, other.city)) {
            return false;
        }
        if (!Objects.equals(this.country, other.country)) {
            return false;
        }
        return true;
    }
```

```java
    @Override
    public String toString() {
        return "Airport{" + "code_AITA=" + code_AITA + ", city=" + city + ", country=" + country + '}';
    }


}
```

4-Création de la classe AirportDAO

```java
import datasave.Airport;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import java.util.ArrayList;

/**
 *
 * @author Formation
 */
public class AirportDAO extends DAO<Airport, String> {

    boolean succed = false;

    public AirportDAO() {

        super();

    }
```

```java
    /*
     cette méthode ajoutera un objet de type airport à la bdd
    elle prend comme argument l'obj à inserrer dans la table
     et retourne ce qui a effectivement été ajouté à la table

     */
    @Override
    public Airport creer(Airport airport) {

        Airport ap = new Airport();

        if (this.bddmanager.connect()) {
            try {
                String query = "INSERT INTO airports VALUES (?, ?, ?)";
                PreparedStatement stInsert = this.bddmanager.getConnectionManager().prepareStatement(query);
                stInsert.setString(1, airport.getCode_AITA().toUpperCase());
                stInsert.setString(2, airport.getCity());
                stInsert.setString(3, airport.getCountry());

                System.out.println(stInsert.toString());

                stInsert.executeUpdate();

                ap = this.find(airport.getCode_AITA());

            } catch (SQLException ex) {
                ex.printStackTrace();
            }
            return ap;

        }
        return ap;
    }
```

```java
/*

Cette méthode prend en argument l'obj a supprimer de la table
(créé avec le constructeur contenant uniquement la clef primaire)
Elle retourne un boolean si la suppression a été effectué ou non

 */
@Override
public boolean supprimer(String obj) {

    String airportKey = obj;

    if (this.bddmanager.connect()) {

        try {
            String querySuppr = " DELETE FROM airports WHERE aita = ?";
            PreparedStatement stSuppr = this.bddmanager.getConnectionManager().prepareStatement(querySuppr);

            stSuppr.setString(1, airportKey);

            System.out.println(stSuppr.toString());

            stSuppr.executeUpdate();

            succed = true;

        } catch (SQLException ex) {
            ex.printStackTrace();

            return succed;
        }

    }

    return succed;
}
```

```java
/*

    Cette methode retourne la table entière

 */
@Override
public ArrayList getAll() {
    ArrayList<Airport> airportList = new ArrayList<>();
    if (this.bddmanager.connect()) {
        try {

            Statement st = this.bddmanager.getConnectionManager().createStatement();
            String requete = "SELECT * FROM airports";
            ResultSet rs = st.executeQuery(requete);

            while (rs.next()) {

                Airport ap = new Airport(rs.getString("aita"), rs.getString("city"), rs.getString("pays"));
                airportList.add(ap);
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            return airportList;
        }

    }
    return airportList;
}
```

```java
/*

Cette méthode prend en argument l'id de la ligne a changer et un objet de
type aéroport qui est les modif à apporter

 */
@Override
public Airport update(String id, Airport obj) {

    Airport ap = (Airport) obj;

    if (this.bddmanager.connect()) {
        try {
            String query = " UPDATE airports SET aita =? , city=?, pays =? where aita=?";
            PreparedStatement stUpdate = this.bddmanager.getConnectionManager().prepareStatement(query);
            stUpdate.setString(1, ap.getCode_AITA());
            stUpdate.setString(2, ap.getCity());
            stUpdate.setString(3, ap.getCountry());
            stUpdate.setString(4, id);

            System.out.println(stUpdate.toString());

            stUpdate.executeUpdate();

        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        return ap;

    }
    return ap;
}
```

```java
/*

Cette méthode retourne l'aéroport avec le code AITA donné en argument

 */
@Override
public Airport find(String id) {

    Airport airport = new Airport();

    if (this.bddmanager.connect()) {
        try {
            String requete = "SELECT * FROM airports WHERE aita = ?";

            PreparedStatement stFind = this.bddmanager.getConnectionManager().prepareStatement(requete);

            stFind.setString(1, id);

            ResultSet rs = stFind.executeQuery();
            if(rs.next()) {
                airport = new Airport(rs.getString("aita"), rs.getString("city"), rs.getString("pays"));
            }


        } catch (SQLException ex) {
            ex.printStackTrace();

        }

    }
    return airport;
}
```

5-Tests Unitaires

```java
/**
 * Test of creer method, of class AirportDAO.
 */
@Test
public void testCreer() {
    System.out.println("creer");
    Airport obj = new Airport("KKK","Machin Lake","USA");

    AirportDAO instance = new AirportDAO();


    Airport expResult = obj;
    Airport result = instance.creer(obj);


    String expectedResult = expResult.toString();
    String resultat = result.toString();

    System.out.println(expectedResult);

    assertEquals(expectedResult, resultat);

}
```

```java
    /**
 * Test of find method, of class AirportDAO.
 */
@Test
public void testFind() {
    System.out.println("find");
    String id = "KKK";
    AirportDAO instance = new AirportDAO();
    String expResult = "Airport{code_AITA=KKK, city=Machin Lake, country=USA}";
    Airport result = instance.find(id);

    String resultat = result.toString();

    assertEquals(expResult, resultat);

}
```

```java
/**
 * Test of supprimer method, of class AirportDAO.
 */
@Test
public void testSupprimer() {
    System.out.println("supprimer");
    String id = "KKK";
    AirportDAO instance = new AirportDAO();
    boolean expResult = true;
    boolean result = instance.supprimer(id);
    assertEquals(expResult, result);
}
```

```java
    /**
     * Test of update method, of class AirportDAO.
     */
    @Test
    public void testUpdate() {
        System.out.println("update");
        String id = "KKK";
        Airport obj = new Airport("KKK","Truck Lake","USA");
        AirportDAO instance = new AirportDAO();
        Airport expResult = obj;
        Airport result = instance.update(id, obj);
        assertEquals(expResult, result);

    }
```

```java
    /**
     * Test of getAll method, of class AirportDAO.
     */
    @Test
    public void testGetAll() {

     System.out.println("get all");
     String result = "";
     String expResult = "";
     AirportDAO instance = new AirportDAO();
     ArrayList<Airport> arrayOfResult = instance.getAll();
        for (Airport airport : arrayOfResult) {
            result += airport.toString();
            expResult += instance.find(airport.getCode_AITA());
        }

     assertEquals(expResult, result);}


}
```

6 DÉBOGAGE

Erreurs les plus courantes

1) Forked Java VM exited abnormally. Please note the time in the report does not reflect the time until the VM exit.
junit.framework.AssertionFailedError
at org.netbeans.core.execution.RunClassThread.run(RunClassThread.java:153)

Solution : Vous n'avez pas ajouter le driver JDBC

1) clique droit sur le projet
2) propriétés
3) Dans l'arbre à gauche sélectionnez Libraries
4) dans la fenêtre à droite cliquez sur "Add Librarie"
5)Ajouter la library "MySQL JDBC Driver"

2) NullPointerExeption

Plusieurs causes sont possible ici , soit la connexion à votre base de données n'est pas faite, soit vous avez été déconnecté, soit votre requête est fausse (généralement le nom des champs)