



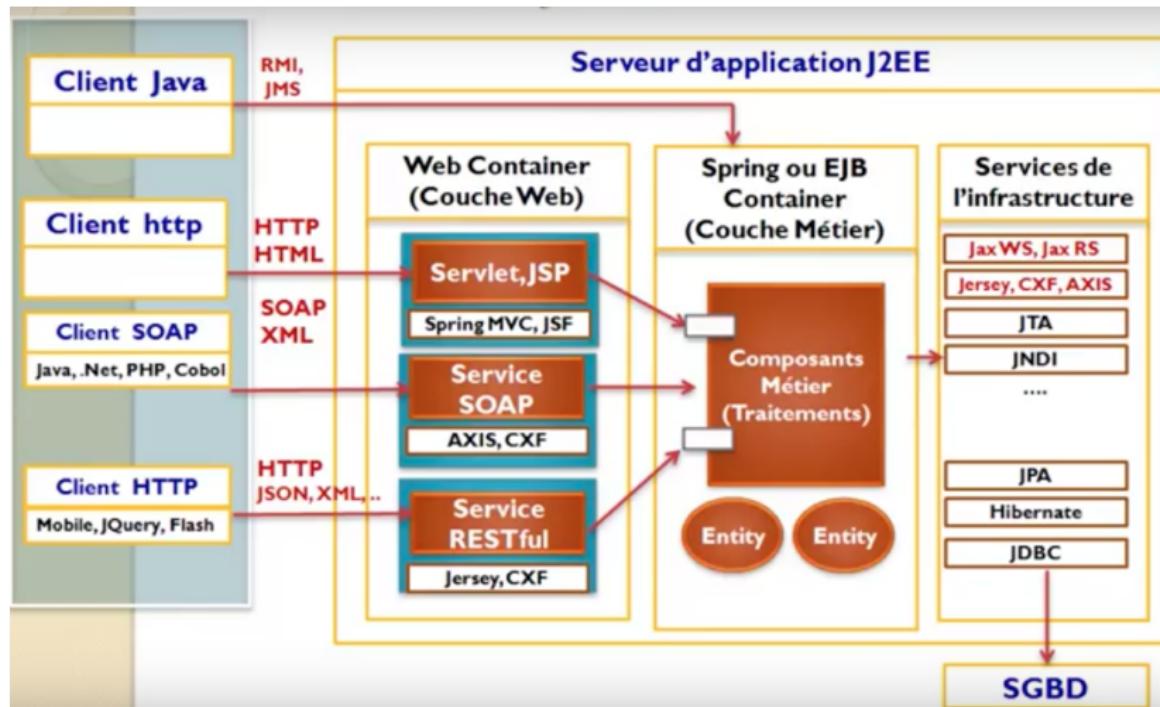
Architecture web (Partie 2)

Yassamine Seladji

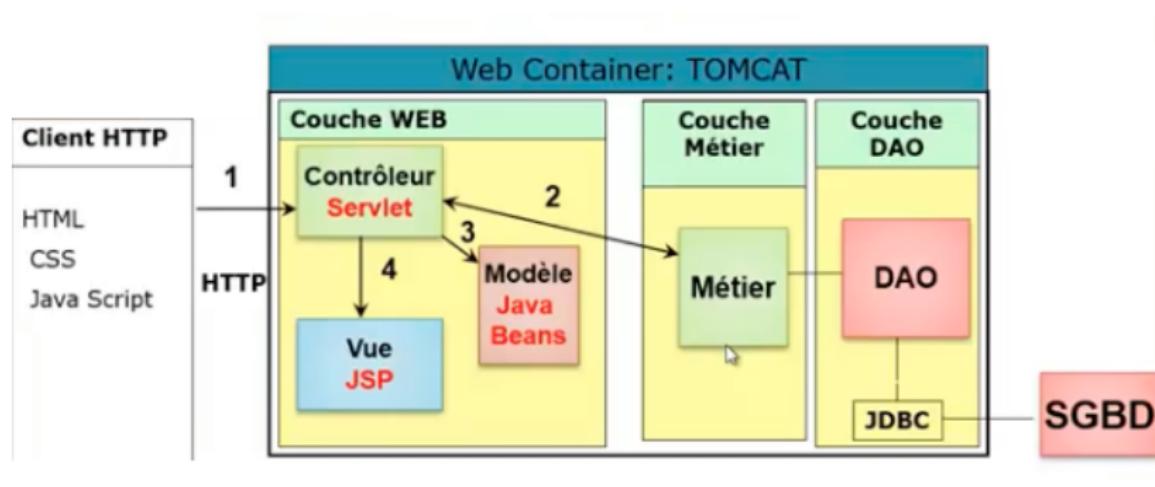
yassamine.seladji@gmail.com

19 avril 2021

Introduction



Introduction



Le mapping Objet/relationnel

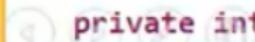
- ▶ La relation entre les objets (classes) de l'application et les champs (tables) en base de données.

Le mapping Objet/relationnel

- ▶ La relation entre les objets (classes) de l'application et les champs (tables) en base de données.

Application :

```
public class Produit{  
    private Long id;  
    private String designation;  
    private double prix;  
    private int quantite;
```



Le mapping Objet/relationnel

- ▶ La relation entre les objets (classes) de l'application et les champs (tables) en base de données.

Application :

```
public class Produit{  
    private Long id;  
    private String designation;  
    private double prix;  
    private int quantite;
```

Bases de données :

SGBDR MySQL, BD DB_CAT			
Table PRODUITS			
ID	DESIGNATION	PRIX	QUANTITE
1	Ordi HL 3421	980	12
2	Imprimante HP LX 7600	2300	10
3	Imprimante Epson HR 450	1300	10

Le mapping Objet/relationnel

```
public List<Produit> produitsParMC(String mc) {
    List<Produit> produits=new ArrayList<Produit>();
    Class.forName("com.mysql.jdbc.Driver");
    Connection conn=DriverManager.getConnection
    ("jdbc:mysql://localhost:3306/DB_CAT","root","");
    PreparedStatement ps=conn.prepareStatement("SELECT * FROM
    PRODUITS WHERE DESIGNATION like ?");
    ps.setString(1, mc);
    ResultSet rs=ps.executeQuery();
    while(rs.next()){
        Produit p=new Produit();
        p.setId(rs.getLong("ID"));
        p.setDesignation(rs.getString("DESIGNATION"));
        p.setPrix(rs.getDouble("PRIX"));
        p.setQuantite(rs.getInt("QUANTITE"));
        produits.add(p);
    }
}
```

Le mapping Objet/relationnel

```
public List<Produit> produitsParMC(String mc) {  
    List<Produit> produits = new ArrayList<Produit>();  
    Class.forName("com.mysql.jdbc.Driver");  
    ("jdbc:mysql://localhost:3306/DB_CAT","root","");
    PreparedStatement ps=conn.prepareStatement("SELECT * FROM
    PRODUITS WHERE DESIGNATION like ?");
    ps.setString(1, mc);
    ResultSet rs=ps.executeQuery();
    while(rs.next()){
        Produit p=new Produit();
        p.setId(rs.getLong("ID"));
        p.setDesignation(rs.getString("DESIGNATION"));
        p.setPrix(rs.getDouble("PRIX"));
        p.setQuantite(rs.getInt("QUANTITE"));
        produits.add(p);
    }
}
```

Le mapping Objet/relationnel

```
public List<Produit> produitsParMC(String mc) {
    List<Produit> produits=new ArrayList<Produit>();
    Class.forName("com.mysql.jdbc.Driver");
    Connection conn=DriverManager.getConnection
    ("jdbc:mysql://localhost:3306/DB_CAT", "root", "");
    PreparedStatement ps=conn.prepareStatement("SELECT * FROM
    PRODUITS WHERE DESIGNATION like ?");
    ps.setString(1, mc);
    ResultSet rs=ps.executeQuery();
    while(rs.next()){
        Produit p=new Produit();
        p.setId(rs.getLong("ID"));
        p.setDesignation(rs.getString("DESIGNATION"));
        p.setPrix(rs.getDouble("PRIX"));
        p.setQuantite(rs.getInt("QUANTITE"));
        produits.add(p);
    }
}
```

Le mapping Objet/relationnel

```
public List<Produit> produitsParMC(String mc) {
    List<Produit> produits=new ArrayList<Produit>();
    Class.forName("com.mysql.jdbc.Driver");
    Connection conn=DriverManager.getConnection
    PreparedStatement ps=conn.prepareStatement("SELECT * FROM
    PRODUITS WHERE DESIGNATION like ?");
    ps.setString(1, mc);
    ResultSet rs=ps.executeQuery();
    while(rs.next()){
        Produit p=new Produit();
        p.setId(rs.getLong("ID"));
        p.setDesignation(rs.getString("DESIGNATION"));
        p.setPrix(rs.getDouble("PRIX"));
        p.setQuantite(rs.getInt("QUANTITE"));
        produits.add(p);
    }
}
```

Le mapping Objet/relationnel

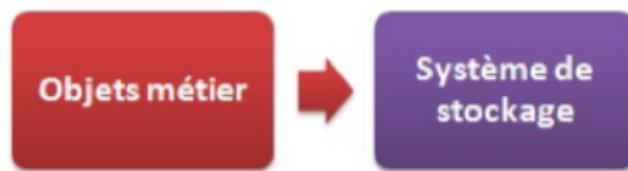
```
public List<Produit> produitsParMC(String mc) {  
    List<Produit> produits=new ArrayList<Produit>();  
    Class.forName("com.mysql.jdbc.Driver");  
    Connection conn=DriverManager.getConnection  
    ("jdbc:mysql://localhost:3306/DB_CAT","root","");
    PreparedStatement ps=conn.prepareStatement("SELECT * FROM  
    PRODUITS WHERE DESIGNATION like ?");  
  
    ResultSet rs=ps.executeQuery();  
    while(rs.next()) {  
        Produit p=new Produit();  
        p.setId(rs.getLong("ID"));  
        p.setDesignation(rs.getString("DESIGNATION"));  
        p.setPrix(rs.getDouble("PRIXT"));  
        p.setQuantite(rs.getInt("QUANTITE"));  
        produits.add(p);  
    }  
}
```

Le mapping Objet/relationnel

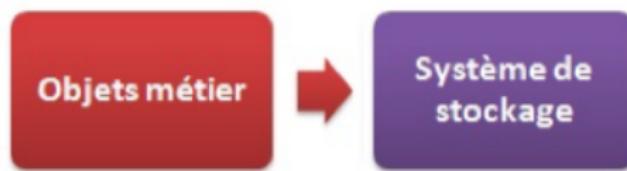
```
public List<Produit> produitsParMC(String mc) {  
    List<Produit> produits=new ArrayList<Produit>();  
    Class.forName("com.mysql.jdbc.Driver");  
    Connection conn=DriverManager.getConnection  
        ("jdbc:mysql://localhost:3306/DB_CAT","root","");
    PreparedStatement ps=conn.prepareStatement("SELECT * FROM  
PRODUITS WHERE DESIGNATION like ?");  
  
    Produit p=new Produit();  
    p.setId(rs.getLong("ID"));  
    p.setDesignation(rs.getString("DESIGNATION"));  
    p.setPrix(rs.getDouble("PRIX"));  
    p.setQuantite(rs.getInt("QUANTITE"));  
    p.setQuantite(rs.getInt("QUANTITE"));  
    produits.add(p);  
}  
}
```

Mapping objet/relationnel

Le mapping Objet/relationnel



Le mapping Objet/relationnel



Les inconvénients :

- ▶ Mélanger le code métier et le code technique (stockage des données).
- ▶ Impossible de mettre en place des tests unitaires séparés.
- ▶ Impossible de changer de mode de stockage.

Le pattern DAO

Isoler le stockage des données.



Le pattern DAO

Isoler le stockage des données.



Le pattern DAO permet de faire la distinction entre les données et leur stockage.

Le pattern DAO

Le pattern DAO définit deux couches :

- ▶ **La couche métier** : gère les traitements métier appliqués aux données.
- ▶ **La couche de stockage des données** : gère les opérations de stockage des données.

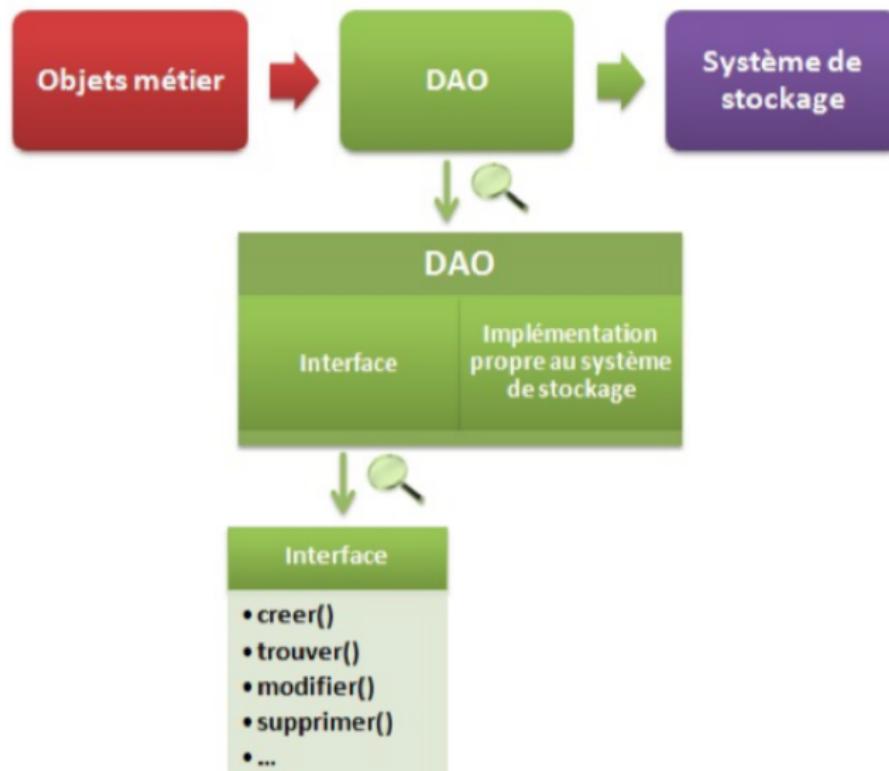
Le pattern DAO

Le pattern DAO définit deux couches :

- ▶ **La couche métier** : gère les traitements métier appliqués aux données.
- ▶ **La couche de stockage des données** : gère les opérations de stockage des données.

La **communication** entre les couches doit se faire via des **interfaces**.

Le pattern DAO

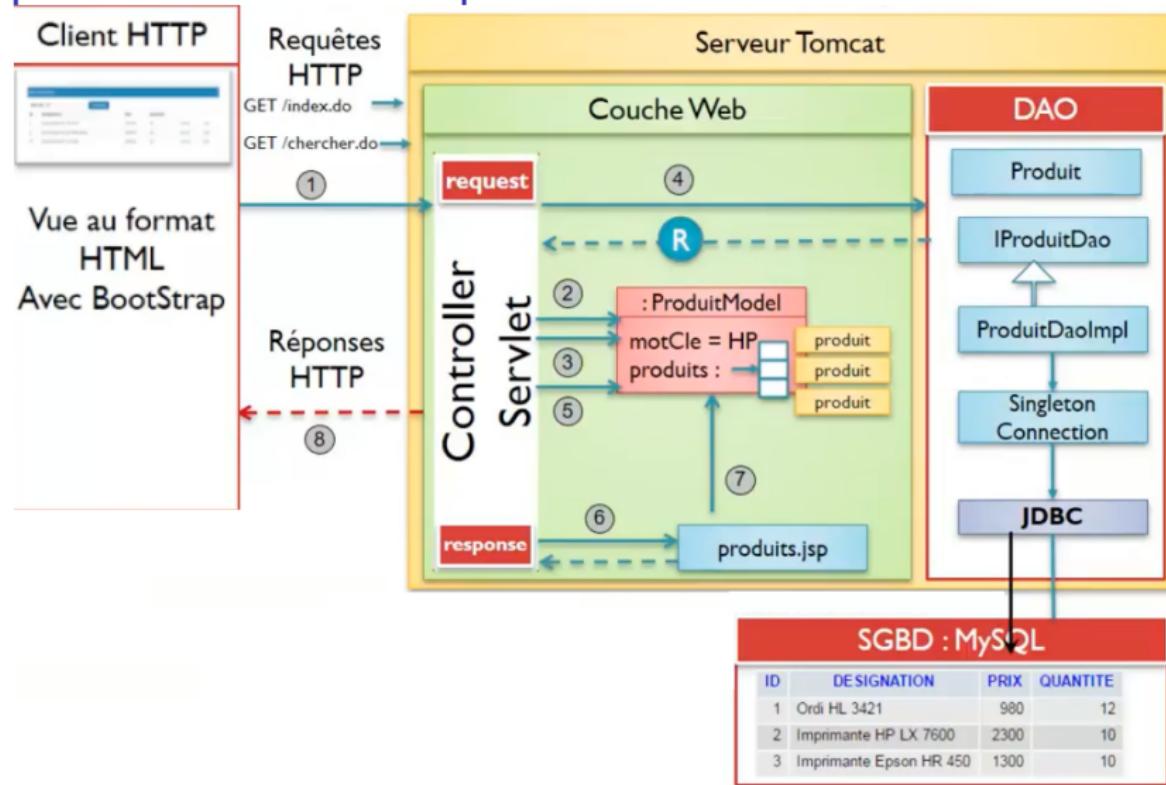


Le pattern DAO

Les étapes de la mise en place de la couche de stockage :

- ▶ La définition de l'interface de communication entre la couche stockage et la couche métier.
- ▶ L'implémentation de l'interface de communication.
- ▶ Définir la classe qui se charge de la connexion à la JDBC (classe Singleton).

Le pattern DAO : exemple



Le pattern DAO : exemple

La classe métier :

```
package metier.entities;

import java.io.Serializable;

public class Produit implements Serializable {
    private Long id; ← Ajouter l'attribut ID
    private String designation;
    private double prix;
    private int quantite;

    // Constructeurs

    public Produit() { }

    public Produit(String designation, double prix, int quantite) {
        this.designation = designation;
        this.prix = prix;
        this.quantite = quantite;
    }

    // Une méthode toString

    @Override
    public String toString() {
        return "Produit [id=" + id + ", designation=" + designation +
               ", prix=" + prix + ", quantite=" + quantite + "]";
    }
}
```

Le pattern DAO : exemple

La classe de la connexion à la JDBC :

```
package dao;
import java.sql.Connection;
import java.sql.DriverManager;
public class SingletonConnection {
    private static Connection connection;
    static{
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection=DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/DB_CATAL","root","");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static Connection getConnection() {
        return connection;
    }
}
```

Le pattern DAO : exemple

La classe de la connexion à la JDBC :

```
package dao;  
import java.sql.Connection;  
import java.sql.DriverManager;  
  
static{  
    try {  
        Class.forName("com.mysql.jdbc.Driver");  
        connection=DriverManager.getConnection  
            ("jdbc:mysql://localhost:3306/DB_CATAL","  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
  
public static Connection getConnection() {  
    return connection;  
}  
}
```

Le pattern DAO : exemple

La classe de la connexion à la JDBC :

```
package dao;
import java.sql.Connection;
import java.sql.DriverManager;
public class SingletonConnection {
    private static Connection connection;
    static{
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection=DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/DB_CATAL","root","");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static Connection getConnection() {
        return connection;
    }
}
```

Le pattern DAO : exemple

L'interface de communication :

```
package dao;  
  
import java.util.List;  
  
import metier.entities.Produit;  
  
public interface IProduitDao {  
    public Produit save(Produit p);  
    public List<Produit> produitsParMC(String mc);  
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
package dao;  
  
import java.util.List; import metier.entities.Produit;  
  
public class ProduitDaoImpl implements IProduitDao {  
  
    @Override  
  
    public Produit save(Produit p) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
    @Override  
  
    public List<Produit> produitsParMC(String mc) {  
        // TODO Auto-generated method stub  
        return null;
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
public List<Produit> produitsParMC(String mc) {
    List< Produit> produits=new ArrayList<Produit>();
    Connection connection=SingletonConnection.getConnection();
    try {
        PreparedStatement ps=connection.prepareStatement
        ("SELECT * FROM PRODUITS WHERE DESIGNATION LIKE ?");
        ps.setString(1, mc);
        ResultSet rs=ps.executeQuery();
        while(rs.next()){
            Produit p=new Produit();
            p.setId(rs.getLong("ID"));
            p.setDesignation(rs.getString("DESIGNATION"));
            p.setPrix(rs.getDouble("PRIX"));
            p.setQuantite(rs.getInt("QUANTITE"));
            produits.add(p);
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return produits;
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
public List<Produit> produitsParMC(String mc) {
    List< Produit> produits=new ArrayList<Produit>();
    connection=SingletonConnection.getConnection()
    try {
        PreparedStatement ps=connection.prepareStatement
        ("SELECT * FROM PRODUITS WHERE DESIGNATION LIKE ?");
        ps.setString(1, mc);
        ResultSet rs=ps.executeQuery();
        while(rs.next()){
            Produit p=new Produit();
            p.setId(rs.getLong("ID"));
            p.setDesignation(rs.getString("DESIGNATION"));
            p.setPrix(rs.getDouble("PRIX"));
            p.setQuantite(rs.getInt("QUANTITE"));
            produits.add(p);
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return produits;
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
public List<Produit> produitsParMC(String mc) {  
    List< Produit> produits=new ArrayList<Produit>();  
    Connection connection=SingletonConnection.getConnection();  
  
    PreparedStatement ps=connection.prepareStatement(  
        "SELECT * FROM PRODUITS WHERE DESIGNATION LIKE ?");  
    ps.setString(1, mc);  
    ResultSet rs=ps.executeQuery();  
  
    while(rs.next()){  
        Produit p=new Produit();  
        p.setId(rs.getLong("ID"));  
        p.setDesignation(rs.getString("DESIGNATION"));  
        p.setPrix(rs.getDouble("PRIX"));  
        p.setQuantite(rs.getInt("QUANTITE"));  
        produits.add(p);  
    }  
    } catch (SQLException e) { e.printStackTrace(); }  
    return produits;  
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
public List<Produit> produitsParMC(String mc) {
    List< Produit> produits=new ArrayList<Produit>();
    Connection connection=SingletonConnection.getConnection();
    try {
        PreparedStatement ps=connection.prepareStatement
        ("SELECT * FROM PRODUITS WHERE DESIGNATION LIKE ?");
        ps.setString(1, mc);
        ResultSet rs=ps.executeQuery();
        Produit p=new Produit();
        p.setId(rs.getLong("ID"));
        p.setDesignation(rs.getString("DESIGNATION"));
        p.setPrix(rs.getDouble("PRIX"));
        p.setQuantite(rs.getInt("QUANTITE"));
        produits.add(p);
    }
} catch (SQLException e) { e.printStackTrace(); }
return produits;
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
public Produit save(Produit p) {
    Connection connection=SingletonConnection.getConnection();
    try {
        PreparedStatement ps=connection.prepareStatement
("INSERT INTO PRODUITS (DESIGNATION,PRIX,QUANTITE) VALUES (?,?,?,?)");
        ps.setString(1, p.getDesignation());
        ps.setDouble(2,p.getPrix());
        ps.setInt(3, p.getQuantite());
        ps.executeUpdate(); // Insertion du produit dans la table PRODUITS
        // Récupérer le ID du produit qui vient d'être inséré
        PreparedStatement ps2=connection.prepareStatement
("SELECT MAX(ID) AS MAX_ID FROM PRODUITS");
        ResultSet rs=ps2.executeQuery();
        if(rs.next()){
            p.setId(rs.getLong("MAX_ID")); // définir le id du produit inséré
        }
        ps.close();
    } catch (SQLException e) { e.printStackTrace(); }
    return p;
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
public Produit save(Produit p) {
    Connection connection=SingletonConnection.getConnection();
    try {
        PreparedStatement ps=connection.prepareStatement
        ("INSERT INTO PRODUITS (DESIGNATION,PRIX,QUANTITE) VALUES (?,?,?,?)");
        ps.setString(1, p.getDesignation());
        ps.setDouble(2,p.getPrix());
        ps.setInt(3, p.getQuantite());
        ps.executeUpdate(); // Insertion du produit dans la table PRODUITS
        // Récupérer le ID du produit qui vient d'être inséré
        PreparedStatement ps2=connection.prepareStatement
        ("SELECT MAX(ID) AS MAX_ID FROM PRODUITS");
        ResultSet rs=ps2.executeQuery();
        if(rs.next()){
            p.setId(rs.getLong("MAX_ID")); // définir le id du produit inséré
        }
        ps.close();
    } catch (SQLException e) { e.printStackTrace(); }
    return p;
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

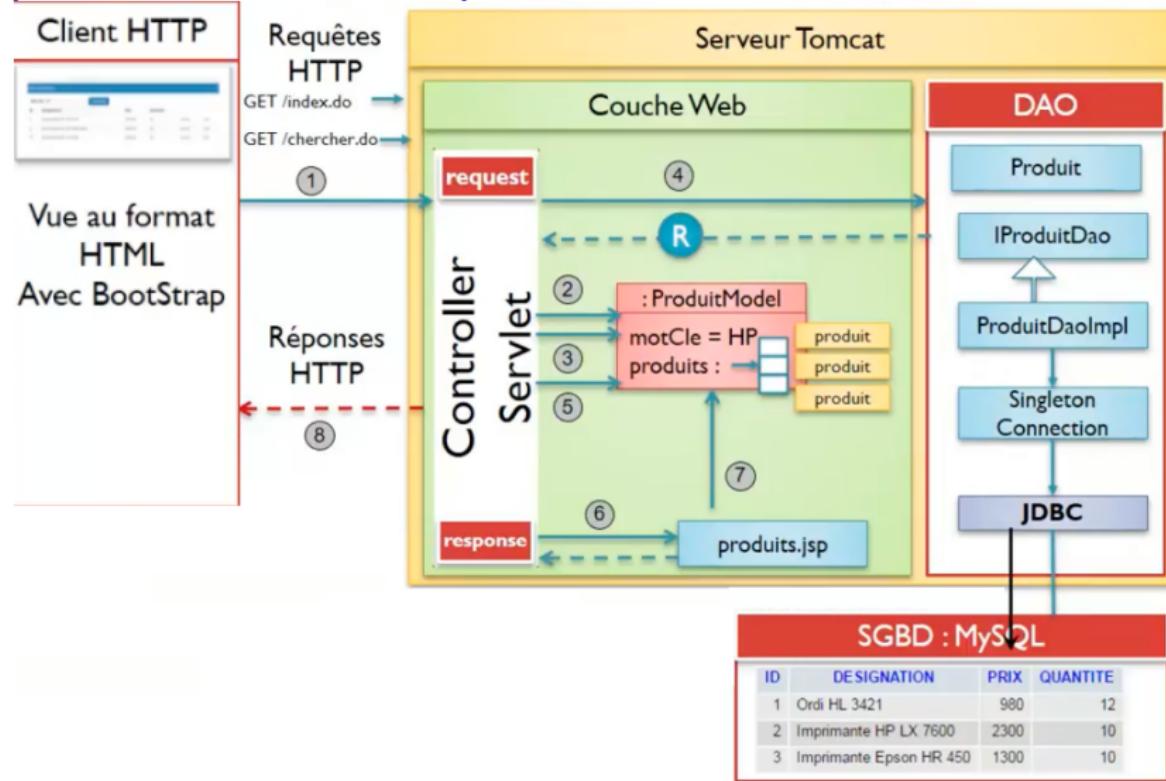
```
public Produit save(Produit p) {
    Connection connection=SingletonConnection.getConnection();
    try {
        PreparedStatement ps=connection.prepareStatement(
            "INSERT INTO PRODUITS (DESIGNATION,PRIX,QUANTITE) VALUES (?)");
        ps.setString(1, p.getDesignation());
        ps.setDouble(2,p.getPrix());
        ps.setInt(3, p.getQuantite());
        ps.executeUpdate(); // Insertion du produit dans la ta
        ("SELECT MAX(ID) AS MAX_ID FROM PRODUITS");
        ResultSet rs=ps.executeQuery();
        if(rs.next()){
            p.setId(rs.getLong("MAX_ID")); // définir le id du produit inséré
        }
        ps.close();
    } catch (SQLException e) { e.printStackTrace(); }
    return p;
}
```

Le pattern DAO : exemple

La classe de l'implémentation de l'interface de communication :

```
public Produit save(Produit p) {
    Connection connection=SingletonConnection.getConnection();
    try {
        PreparedStatement ps=connection.prepareStatement
("INSERT INTO PRODUITS (DESIGNATION,PRIX,QUANTITE) VALUES (?,?,?)");
        ps.setString(1, p.getDesignation());
        ps.setDouble(2,p.getPrix());
        PreparedStatement ps2=connection.prepareStatement
"SELECT MAX(ID) AS MAX_ID FROM PRODUITS";
        ResultSet rs=ps2.executeQuery();
        if(rs.next()){
            p.setId(rs.getLong("MAX_ID")); // définir le id du produit
        }
        ps.close();
    } catch (SQLException e) { e.printStackTrace(); }
    return p;
}
```

Le pattern DAO : exemple



Le pattern DAO : exemple

Donner l'implémentation de la couche communication constituée de :

- ▶ La classe modèle
- ▶ Le contrôleur : la servlet.
- ▶ La vue : la JSP.

Le pattern DAO : exemple

La classe modèle :

```
package web;
import java.util.ArrayList;
import java.util.List;
import metier.entities.Produit;
public class ProduitModel {
    private String motCle="";
    private List<Produit> produits=new ArrayList<Produit>();
    // Getters et Setters
}
```

Le pattern DAO : exemple

Le contrôleur :

```
package web;
import ...;
@WebServlet(name="cs",urlPatterns={"*.do"})
public class ControleurServlet extends HttpServlet{
    private IProduitDao metier;
    @Override
    public void init() throws ServletException {
        metier=new ProduitDaoImpl();
    }
    @Override
    protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
        String path=request.getServletPath();
        if(path.equals("/index.do")){
            request.getRequestDispatcher("produits.jsp").forward(request, response);
        }
    }
}
```

Le pattern DAO : exemple

Le contrôleur :

```
package web;
import ...;
@WebServlet(name="cs",urlPatterns={"*.do"})
public class ControleurServlet extends HttpServlet{
    private IProduitDao metier;
    @Override
    public void init() throws ServletException {
        metier=new ProduitDaoImpl();
    }
    protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, I
    String path=request.getServletPath();
    if(path.equals("/index.do")){
        request.getRequestDispatcher("produits.jsp").forward(request, response);
    }
}
```

Le pattern DAO : exemple

Le contrôleur :

```
package web;  
import ...;  
@WebServlet(name="cs",urlPatterns={"*.do"})  
public class ControleurServlet extends HttpServlet{  
    private IProduitDao metier;  
    @Override  
    public void init() throws ServletException {  
        metier=new ProduitDaoImpl();  
    }  
    @Override  
  
    String path=request.getServletPath();  
    if(path.equals("/index.do")){  
        request.getRequestDispatcher("produits.jsp").forward(  
    }
```

Le pattern DAO : exemple

Le contrôleur :

```
else if(path.equals("/chercher.do")){
    String motCle=request.getParameter("motCle");
    ProduitModel model=new ProduitModel();
    model.setMotCle(motCle);
    List<Produit> produits=metier.produitsParMC("%"+motCle+"%");
    model.setProduits(produits);
    request.setAttribute("model", model);
    request.getRequestDispatcher("produits.jsp").forward(request, response);
}
else{
    response.sendError(Response.SC_NOT_FOUND);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    doGet(request, response);
}
```

Le pattern DAO : exemple

Le contrôleur :

```
else if(path.equals("/chercher.do")){
    ProduitModel model=new ProduitModel();
    model.setMotCle(motCle);
    List<Produit> produits=metier.produitsParMC("%"+motCle+"%");
    model.setProduits(produits);
    request.setAttribute("model", model);
    request.getRequestDispatcher("produits.jsp").forward(request, response);
}
else{
    response.sendError(Response.SC_NOT_FOUND);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    doGet(request, response);
}
```

Le pattern DAO : exemple

Le contrôleur :

```
else if(path.equals("/chercher.do")){
    String motCle=request.getParameter("motCle");
    ProduitModel model=new ProduitModel();
    model.setMotCle(motCle);
    List<Produit> produits=metier.produitsParMC("%"+motCle+"%");
    model.setProduits(produits);

    getRequestDispatcher("produits.jsp").forward(
}

else{
    response.sendError(Response.SC_NOT_FOUND);
}
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    doGet(request, response);
}
```

Le pattern DAO : exemple

La vue :

```
<%@page import="metier.entities.Produit"%> <%@page import="web.ProduitModel"%>
<%
    ProduitModel model;
    if(request.getAttribute("model")!=null)  model=(ProduitModel)request.getAttribute("model");
    else model=new ProduitModel();
%>
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css"/>
</head>
<body>
    <div class="container col-md-10 col-md-offset-1">
        <div class="panel panel-primary">
            <div class="panel-heading">Recherche des produits</div>
            <div class="panel-body">
                <form action="chercher.do" method="get">
                    <label>Mot Clé</label> <input type="text" name="motcle" value="<%model.getMotcle()%>" />
                
```

↳

Le pattern DAO : exemple

La vue :

```
<table class="table table-striped">
    <tr> <th>ID</th><th>Désignation</th><th>Prix</th><th>Quantité</th> </tr>
    <% for(Produit p:model.getProduits()){ %>
        <tr>
            <td><%=p.getId() %></td>
            <td><%=p.getDesignation()%></td>
            <td><%=p.getPrix() %></td>
            <td><%=p.getQuantite() %></td>
            <td><a href="Supprime.do?id=<%=p.getId()%>">Supprimer</a></td>
            <td><a href="Edit.do?id=<%=p.getId()%>">Edit</a></td>
        </tr>
    <% } %>
</table>
</div>
</div>
</body>
</html>
```

Application

Mettre en place une application Web JEE qui permet d'échanger des cours :

- ▶ Un cours est représenté par son titre, son auteur et son contenu.
- ▶ L'utilisateur peut ajouter ou consulter un cours via une interface HTML.

Application

Mettre en place une application Web JEE qui permet d'échanger des cours :

- ▶ Un cours est représenté par son titre, son auteur et son contenu.
- ▶ L'utilisateur peut ajouter ou consulter un cours via une interface HTML.
- ▶ Donner l'architecture utilisée, en précisant les différentes couches.
- ▶ Donner l'implémentation des différentes couches proposées.