

# Le Développement en Couches, et la Persistance des Données avec Java SE



# Le Développement en Couches ... avec Java SE

**Présentation du Cours**



## Objectifs

- Savoir développer une application client lourd structurée
- Concepts java d'interface, de collection et de généricité
- La notion de couches logicielles
- La cohésion, et le couplage
- Les différentes couches :
  - Graphical User Interface (GUI)
  - Business Logic Layer (BLL),
  - Data Access Layer (DAL)
- Le Modèle Vue Contrôleur (MVC)
- Les API : Swing et JDBC
- Les Designs Patterns DAO, Observer, Factory, Singleton

## Configuration

- Java Developpement Kit 8
- Eclipse Neon
- Base de données SQL SERVER

# Le Développement en Couches ... avec Java SE

**Module 01 – Java Notions Complémentaires**



# Objectifs

- Comprendre et implémenter :
  - les Interfaces
  - les Collections
  - les Génériques

# Définir une *Interface*

[public] interface *NomInterface*

[extends *NomSuperInterface1*,  
*NomSuperInterface2*, ...] {

// constantes

[public][static] [final] type *NOM\_CONSTANTE* = *valeur*;

// méthodes abstraites

[public] [abstract] type *methode*( *liste\_parametres* );

}

# Implémenter une *Interface*

```
public interface Predateur {  
    void chasse(Proie p);  
}
```

```
public class Chat implements Predateur {  
    @Override  
    public void chasse(Proie p) {  
        ...  
    }  
}
```

# L'intérêt de l'*Interface*

- Complémentaire à l'héritage :

- Remplace l'héritage multiple
- Permet de faire partager des fonctionnalités communes à des classes différentes
- Vue comme un contrat imposé aux classes qui l'implémentent
- Vue comme un type

JAVA – Notions complémentaires

# Les *Interfaces*

## Démonstration



## Les Collections

- Besoin de variables faisant référence à plusieurs éléments
- 3 solutions :
  - Utiliser un tableau

```
Personne[] equipe = new Personne[10];
```

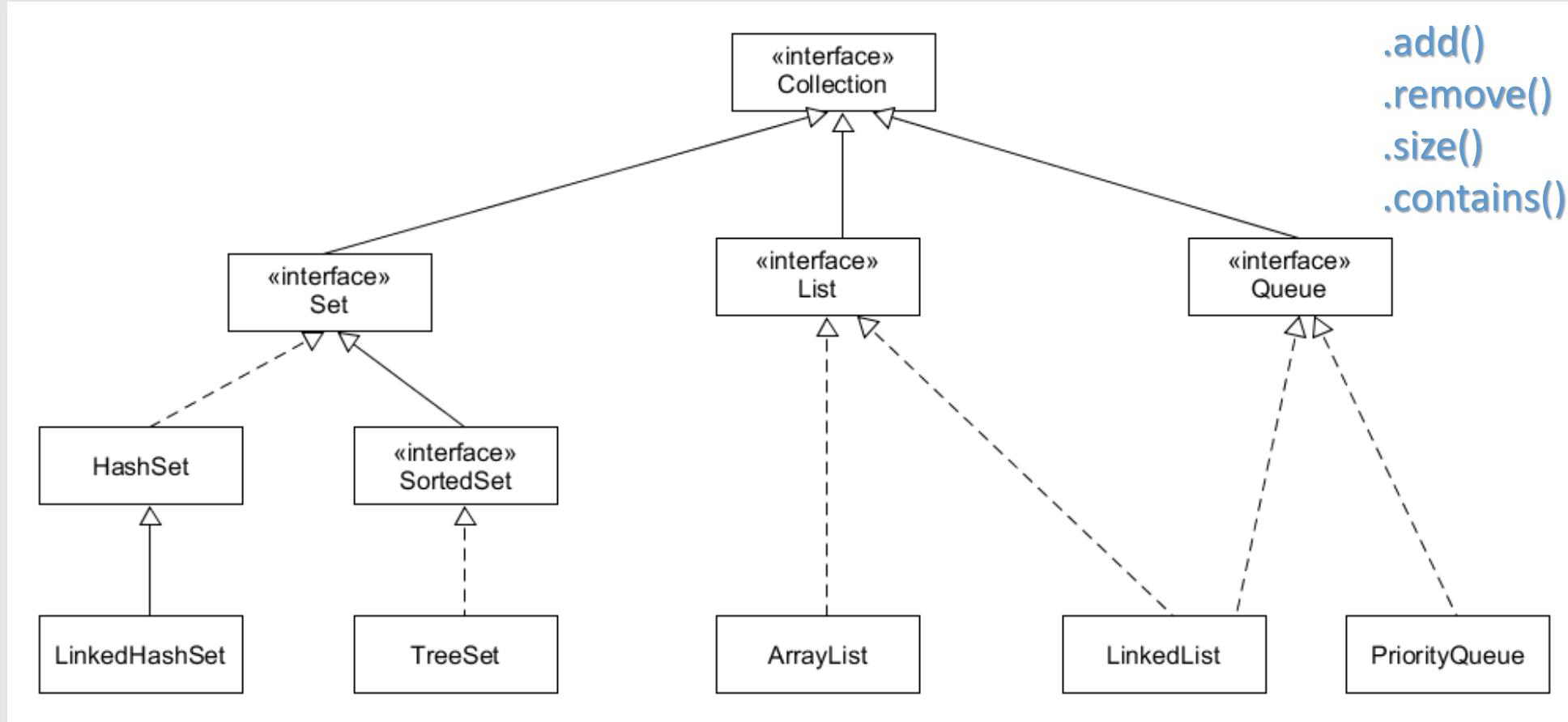
- Utiliser une collection

```
List<Personne> equipe = new ArrayList<Personne>();
```

- Utiliser un dictionnaire clé/valeur

```
Map<Integer, Personne> equipe = new HashMap<>();
```

# Les différents types de *Collections*



JAVA – Notions complémentaires

# Les *Collections*

## Démonstration



# Le dictionnaire Clé/Valeur

```
//Déclarer et instancier le dictionnaire clé/valeur
Map<Integer, Employe> employes = new HashMap<Integer, Employe>();

//Ajouter les éléments
employes.put(emp1.getNoEmploye(), emp1);
employes.put(emp2.getNoEmploye(), emp2);
employes.put(emp3.getNoEmploye(), emp3);

//Accéder aux éléments par leur clé
Employe emp = employes.get(1); // Employé no 1
```

# La Généricité

- Extension au typage fort
- Déetecter les erreurs à la compilation
- Pouvoir passer un type en paramètre
- Syntaxe : utilisation du « diamant » <Type1[,Type2, ...]>

JAVA – Notions complémentaires

# La Généricité

## Démonstration



# Définir et utiliser la Généricité

```
public class MonGenerique<T> {  
    private T t;  
  
    public void add(T t){  
        this.t = t;  
    }  
  
    public T get(){  
        return t;  
    }  
}
```

Type Paramètre

Type Argument

```
public static void main(String[] args) {  
    MonGenerique<String> mc = new MonGenerique<>();  
    mc.add("element");  
    String s = mc.get();  
}
```

# La Généricité : appliquer des contraintes

MaClasse<T extends Vehicule>

MaClasse<T extends Vehicule & Comparable & Cloneable>

UNE Classe

Des Interfaces

Paire<X extends Vehicule, Y extends Personne >

UNE Classe

UNE Classe

MaClasse<T extends Personne & Vehicule>

DES Classes !

MaClasse<T extends List & Vehicule>

# Le Développement en Couches ... avec Java SE

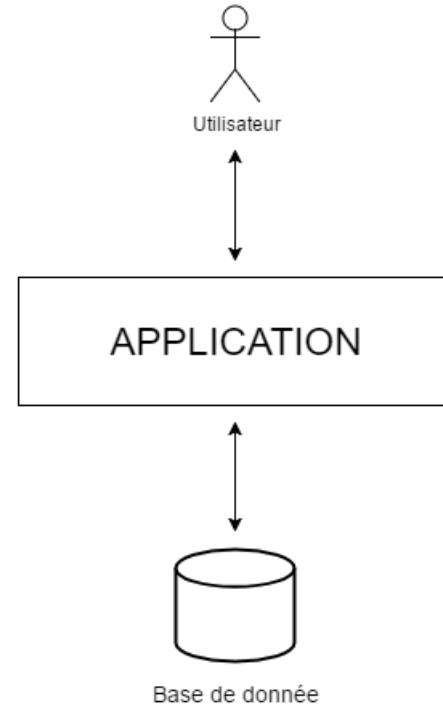
**Module 02 – L'Architecture en Couches**



# Objectifs

- Acquérir une vision globale de l'application
- Réfléchir aux problématiques
- Comprendre la notion de couche logicielle
- Introduire les notions de cohésion et de couplage
- Connaître les différentes façons d'implémenter les couches

# La problématique : une seule couche

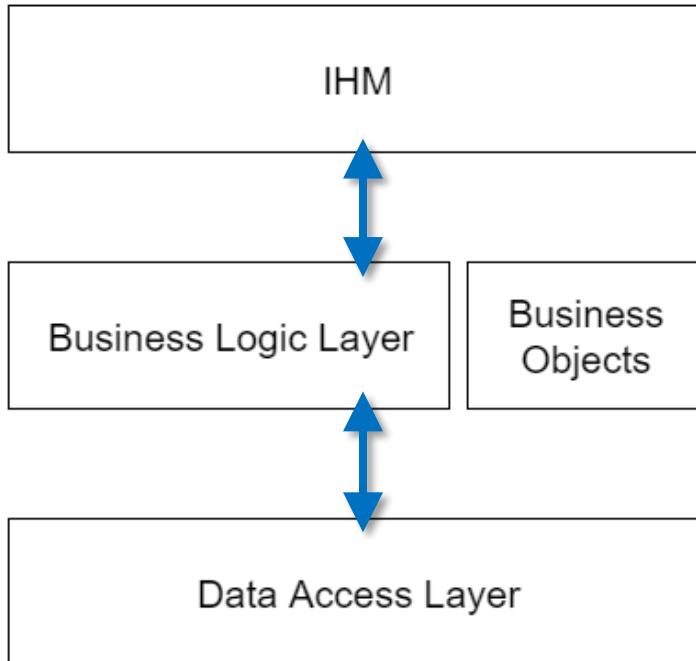


### Inconvénients :

- Les interfaces graphiques, les règles de gestion métier, et persistance des données ...  
... tous ces aspects sont mélangés.
- Maintenance corrective et évolutive difficile
- Réutilisation du code impossible
- Partage du travail compliqué

# L'Architecture en Couches

## La couche Logicielle



### Définition :

« Une couche logicielle est un composant logiciel pouvant communiquer avec ses couches voisines. »

- Cohésion : Une responsabilité par couche
- Couplage : Nature des liens entre couches.
- Permet également :
  - De cacher les niveaux inférieurs d'implémentation
  - Des composants réutilisables et interchangeables

# L'Architecture en Couches

# Implémenter des couches en packages

## Démonstration



# L'Architecture en Couches Implémenter des couches en projets

## Démonstration



L'Architecture en Couches

Implémenter des couches en fichiers .jar

# Démonstration



# Le Développement en Couches ... avec Java SE

**Module 03 – Le Développement de la Couche  
Business Objects (BO)**

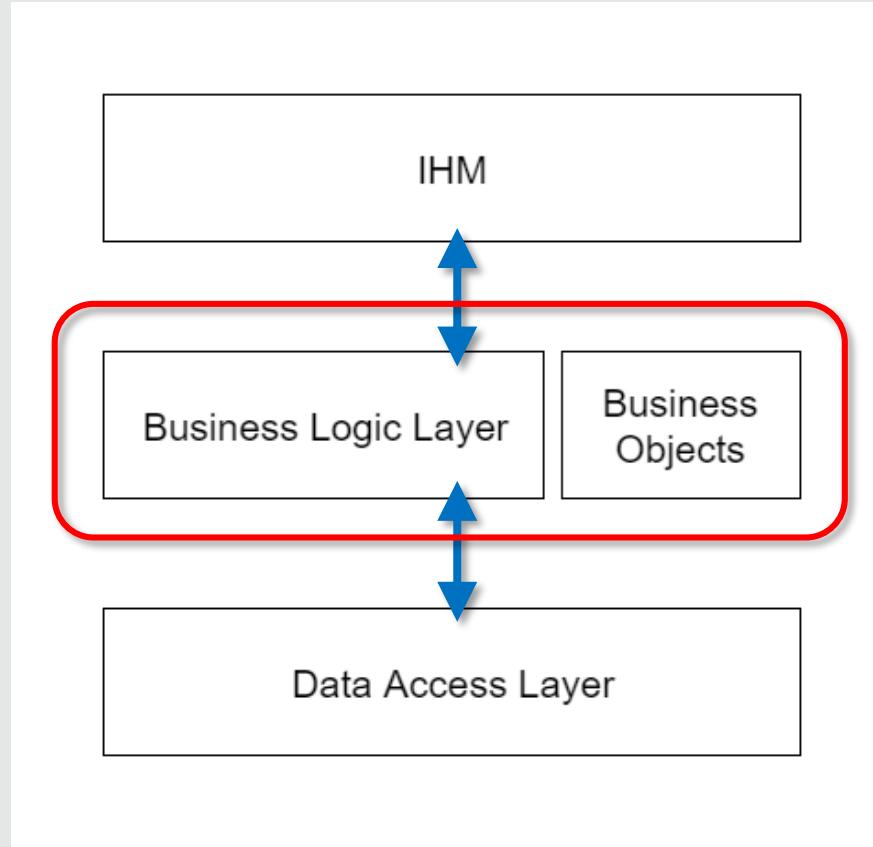


## Objectifs

- Situer la couche Business Objects (BO)
- Connaître les responsabilités de la couche BO
- Comprendre les concepts liés à la couche BO
- Savoir implémenter la couche BO
  - Utiliser l'encapsulation
  - Utiliser les associations entre classes
  - Utiliser l'héritage

Le Développement de la couche Business Objects (BO)

# Situer la couche Business Objects (BO)



# Le Développement de la couche Business Objects (BO)

## Construire la couche Business Objects

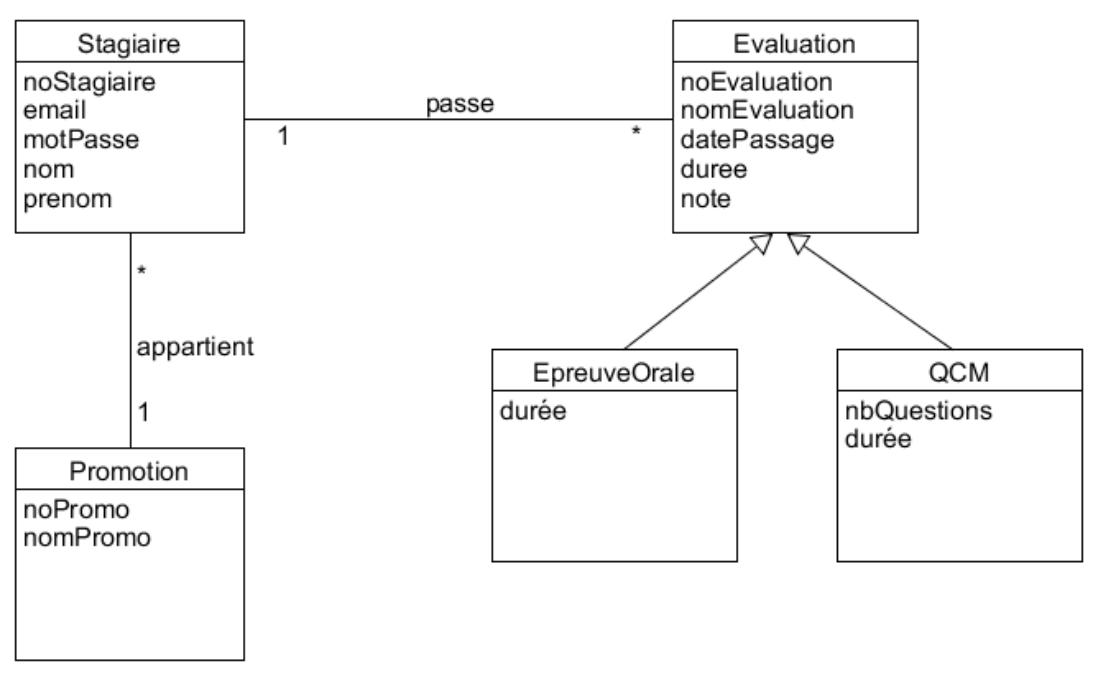


Diagramme de classe – Evaluation stagiaire

- Modèle de données métier :
  - Classes Simples ou POJO (*Plain Old Java Objects*)
  - Associations entre classes :
    - 1:1
    - 1:n
    - n:m
  - Héritage de classe
  - Implémentation d'interface

# Le Développement de la couche Business Objects (BO)

## Utiliser des classes simples

### Structure d'une classe :

- Encapsulation des données :
  - Attributs privés
- Constructeurs :
  - Constructeur vide
  - Constructeur avec paramètres
- Accesseurs (publics) :
  - Getters
  - Setters

```
package fr.eni.evaluations.bo;

import java.util.List;

public class Stagiaire {
    private int noStagiaire;
    private String prenom, nom, email, motDePasse;

    //Constructeur vide
    public Stagiaire(){
        ...
    }

    //Constructeur avec paramètres
    public Stagiaire(String prenom, String nom, String email, String motDePasse,
                     ...)
}

//Getters et Setters
public int getNoStagiaire() {
    return noStagiaire;
}

public void setNoStagiaire(int noStagiaire) {
    this.noStagiaire = noStagiaire;
}
```

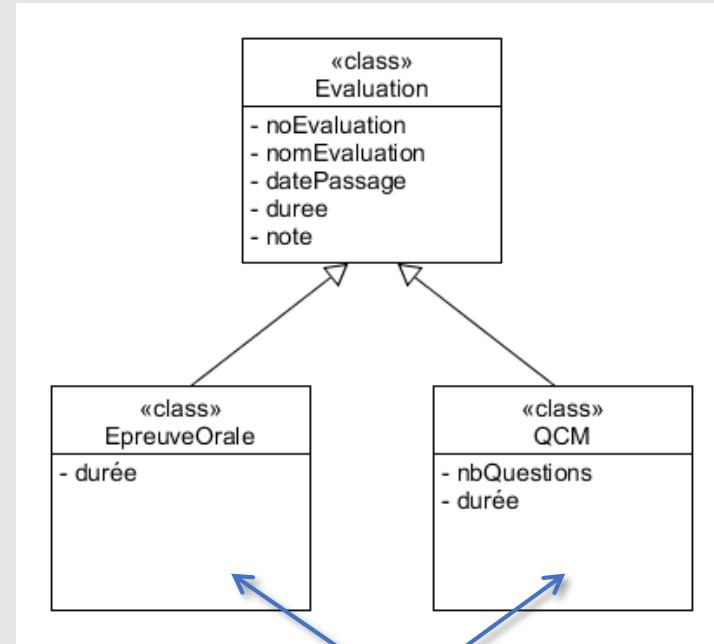
# Le Développement de la couche Business Objects (BO)

## Utiliser l'héritage

Généralisation

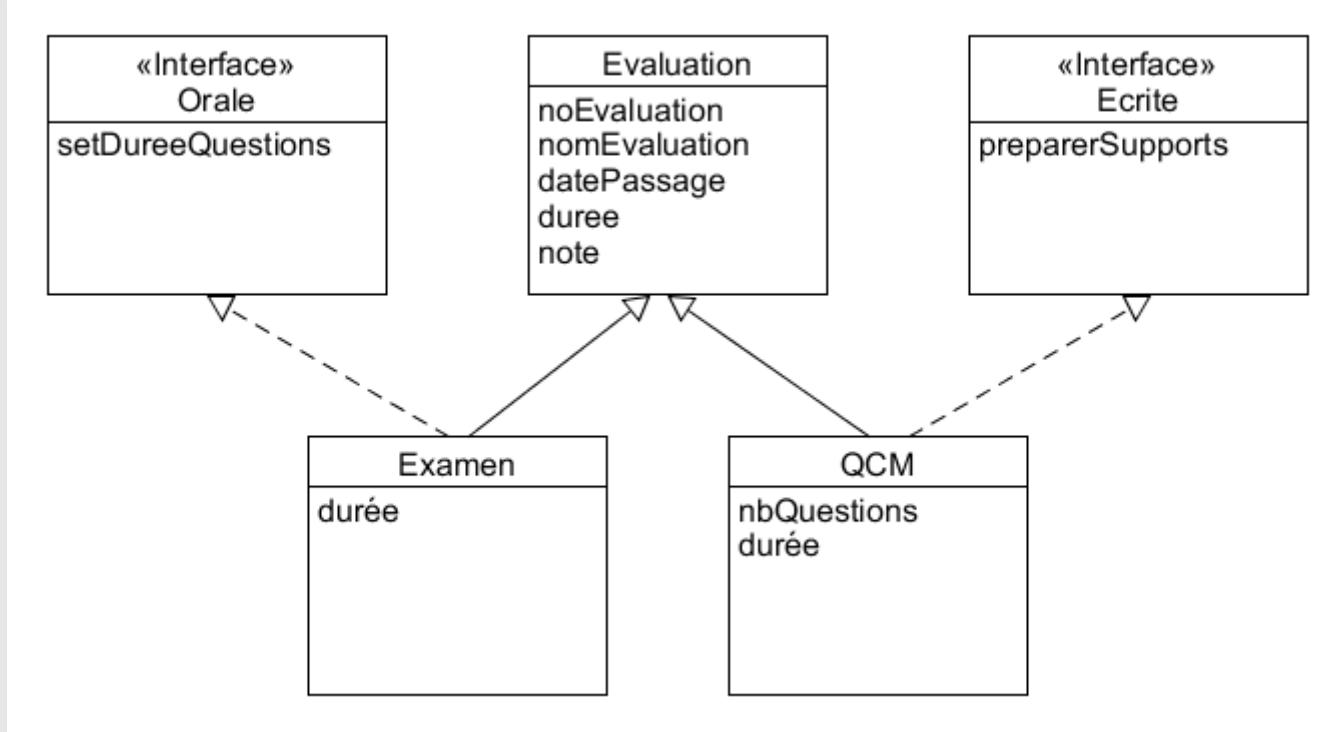
Spécialisation

Catégorisation



# Le Développement de la couche Business Objects (BO)

## Utiliser les interfaces



```
package fr.eni.evaluations.bo;

public interface Ecrite {

    public abstract void setDureeQuestions(int duree);

    public abstract int getDureeQuestions();

}
```

```
package fr.eni.evaluations.bo;

public class Examen extends Evaluation implements Ecrite{

    private int dureeQuestions;

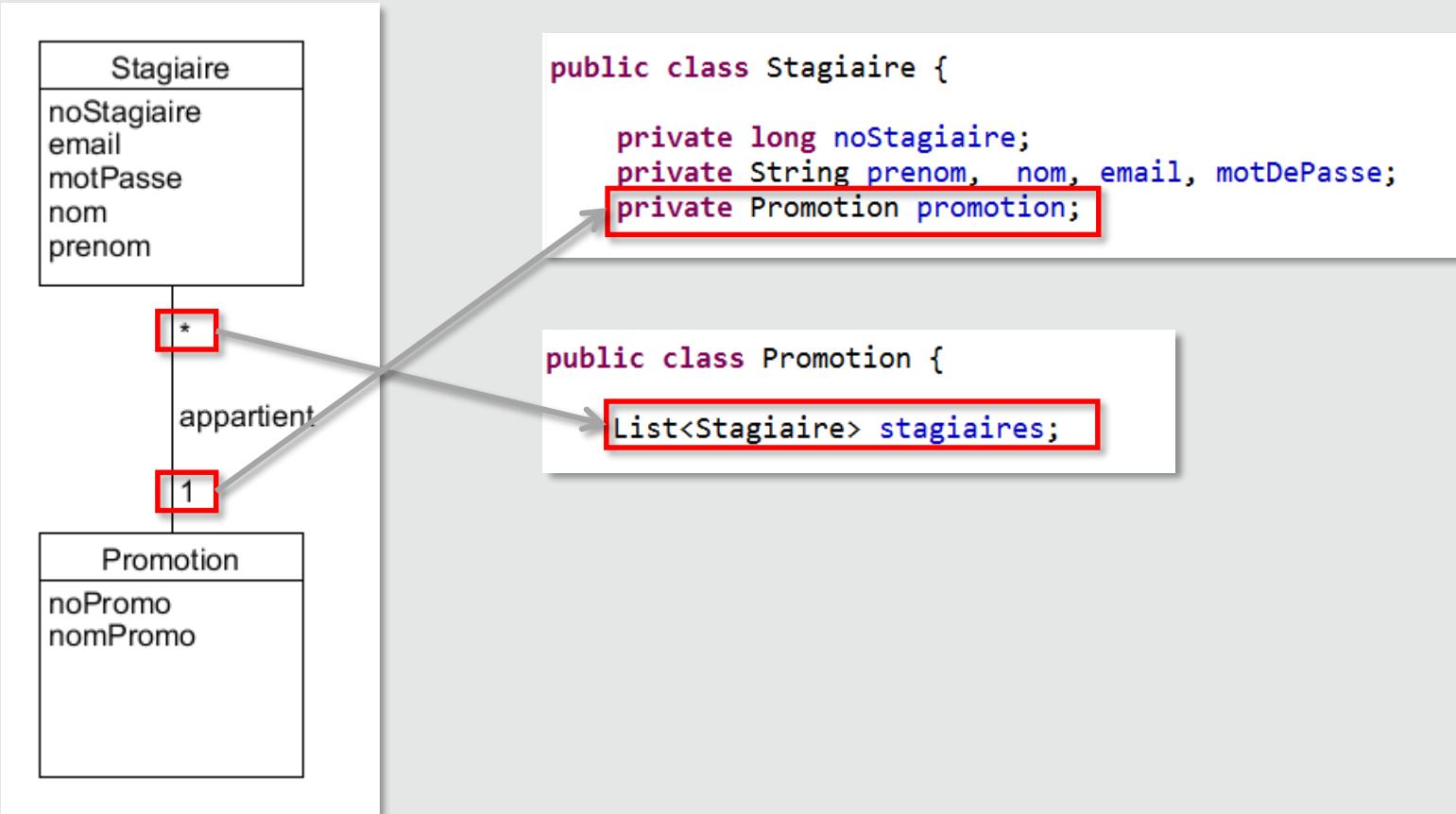
    @Override
    public void setDureeQuestions(int duree) {
        this.dureeQuestions = duree;
    }

    @Override
    public int getDureeQuestions() {
        return dureeQuestions;
    }

}
```

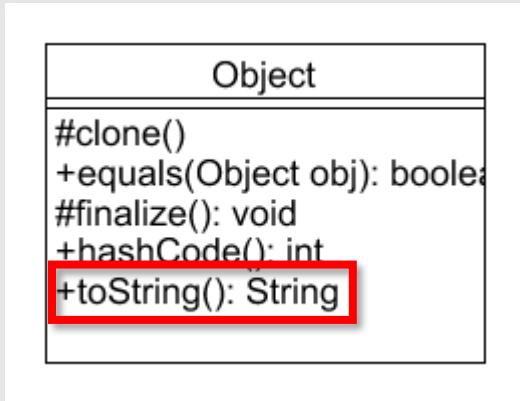
# Le Développement de la couche Business Objects (BO)

## Relier les entités



# Le Développement de la couche Business Objects (BO)

## Implémenter `toString()`



- Donne l'état de l'instance
- Donne de l'information en phase de *debug*
- Exemple d'implémentation :

```
@Override  
public String toString() {  
    StringBuilder sb = new StringBuilder();  
    sb.append("Login [login=");  
    sb.append(login);  
    sb.append(", password=");  
    sb.append(password);  
    sb.append("]");  
    return sb.toString();  
}
```

# Le Développement de la couche Business Objects (BO) Gestion d'une Papeterie - partie 1

## TP

-  Implémenter la couche Business Objects (BO) d'une application Java
-  L'énoncé complet est disponible en téléchargement
-  La durée est estimée à 2 heures

# Le Développement en Couches ... avec Java SE

**Module 04 – Le Développement de la Couche  
Data Access Layer (DAL)**

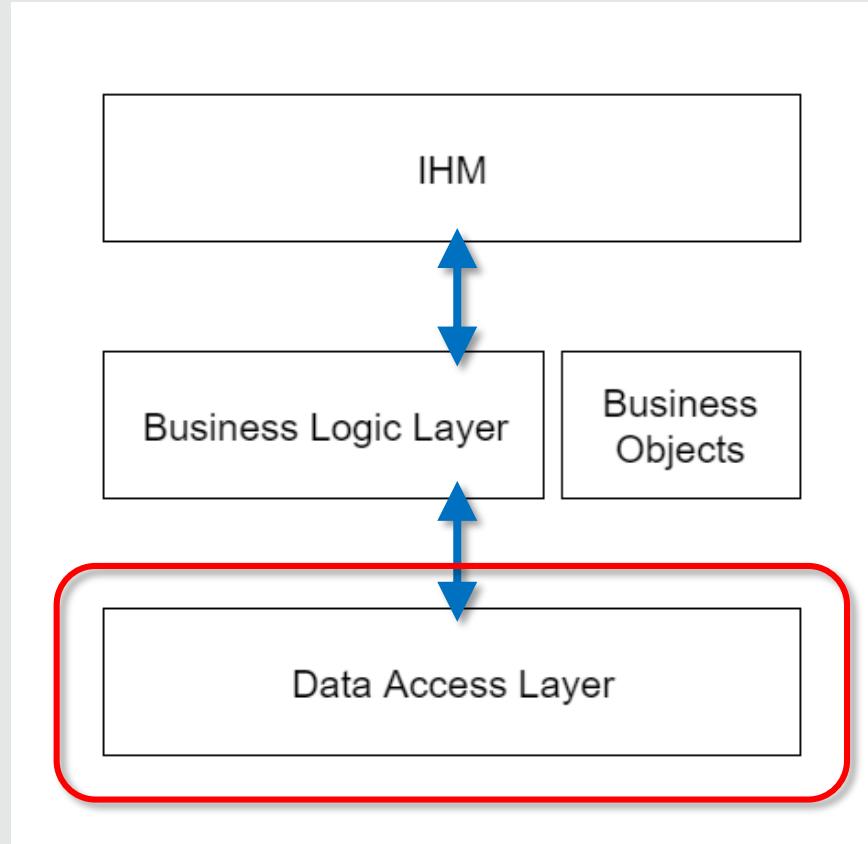


## Objectifs

- Situer la couche DAL
- Responsabilités de la couche DAL
- Comprendre l'organisation de la couche Data Access Layer (DAL)
- Savoir implémenter la couche DAL
- Connaître et utiliser l'API JDBC
- Comprendre le design pattern DAO

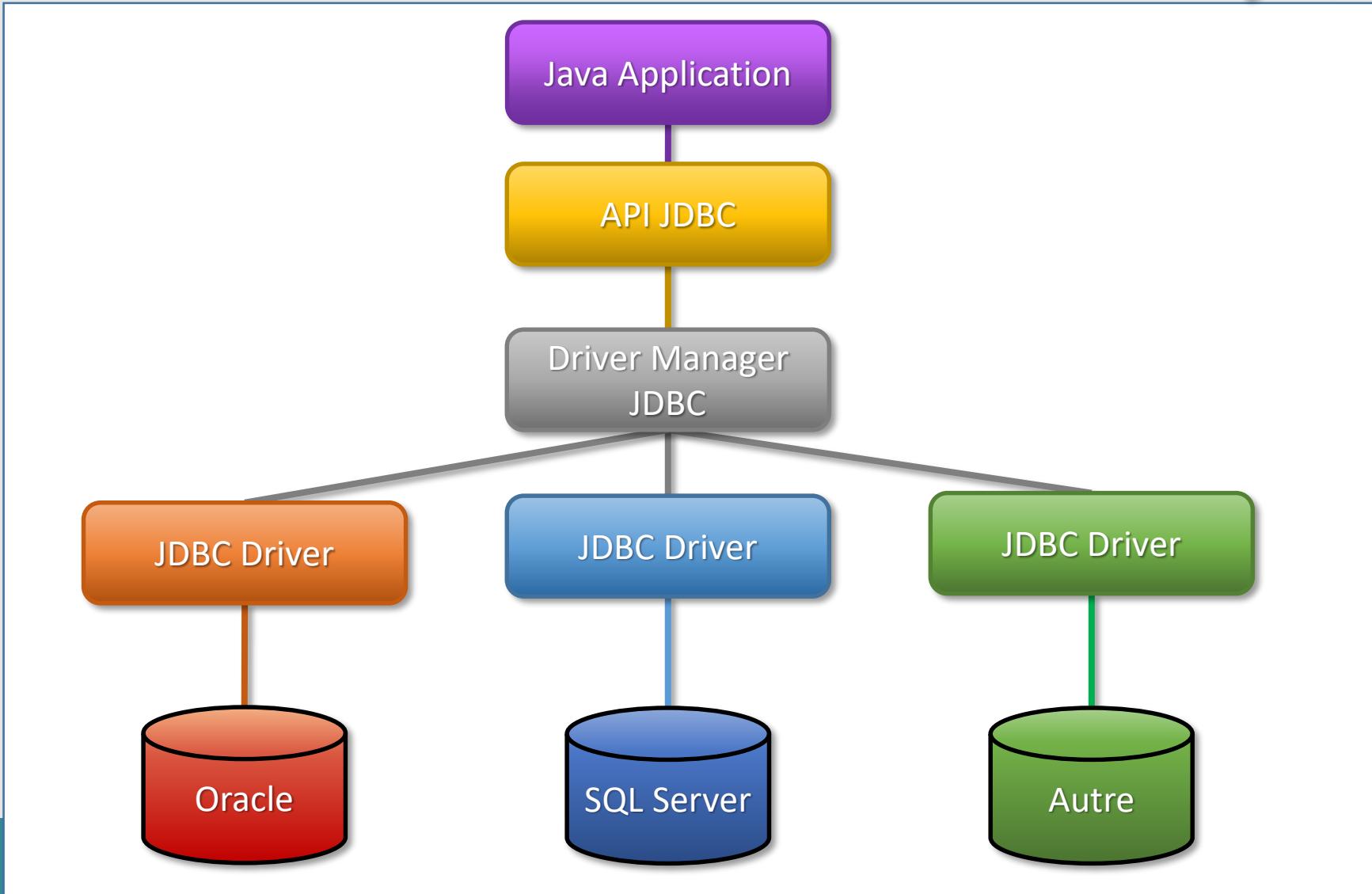
Le Développement de la couche Data Access Layer (DAL)

# Situer la couche Data Access Layer (DAL)



Le Développement de la couche Data Access Layer (DAL)

# L'Architecture Java DataBase Connectivity (JDBC)



# Le Développement de la couche Data Access Layer (DAL)

## Mettre en place l'environnement

- Java est installé
  - JDK pour un poste de développement
  - JRE pour un poste de production
- Une base de données est installée
- Récupérer le pilote correspondant à la base de données installée
- Ajouter le chemin d'accès au pilote dans le « classpath »

Le Développement de la couche Data Access Layer (DAL)  
Mettre en place l'environnement

# Démonstration



# Le Développement de la couche Data Access Layer (DAL)

## Charger le pilote JDBC

- Méthode recommandée par Oracle :

```
DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());
```

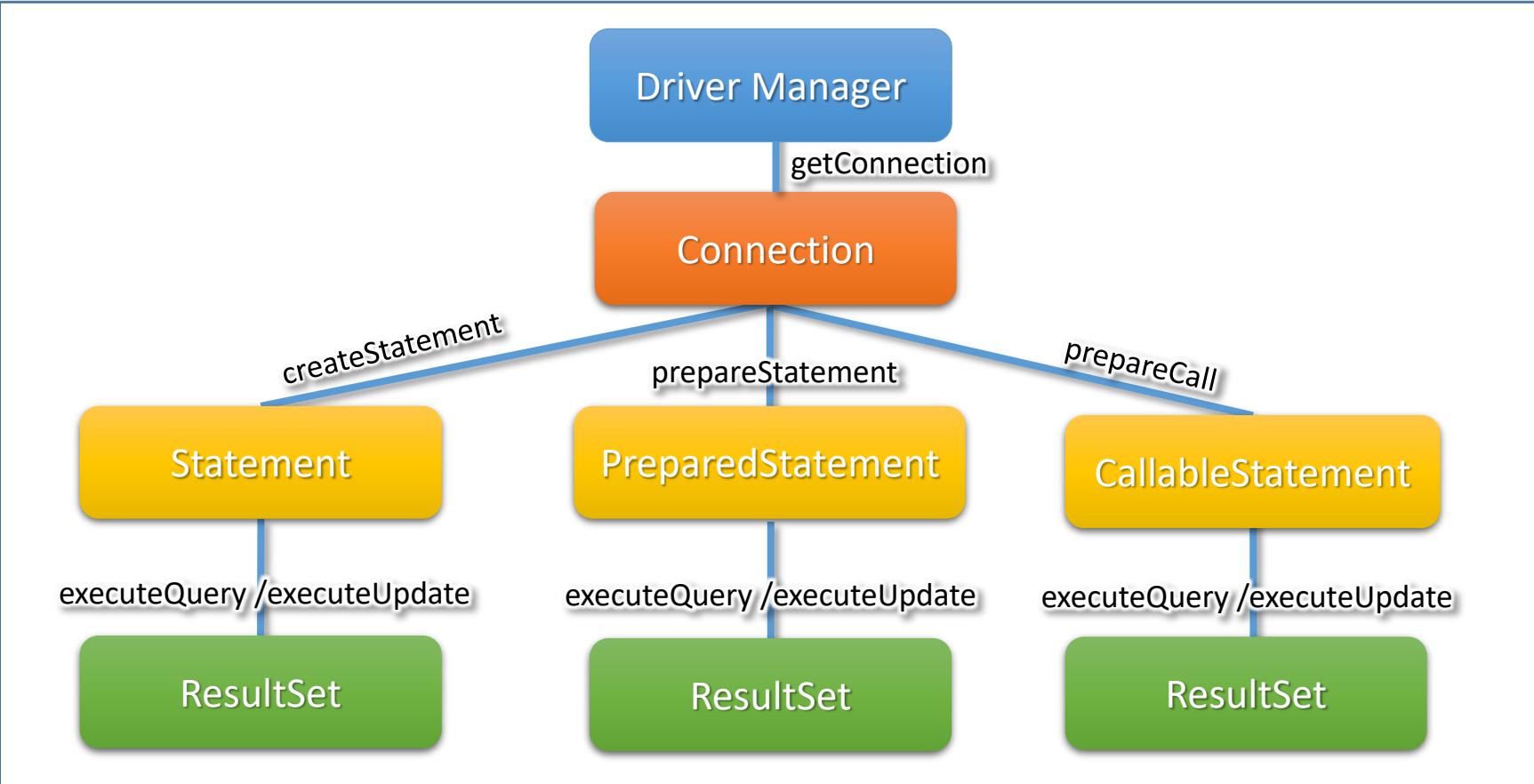
- Méthode évitant la dépendance forte <sup>2</sup>:

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

- Chargement automatique à partir de JDBC v4.

# Le Développement de la couche Data Access Layer (DAL)

## Présentation de l'API JDBC



# Le Développement de la couche Data Access Layer (DAL)

## Établir une Connexion

- Créer une connexion

```
String urlConnection = "jdbc:sqlserver://127.0.0.1;databaseName=TestsJDBC";
Connection connection = DriverManager.getConnection(urlConnection, "sa", "thePasword");
```

- Fermer une connexion

```
connection.close();
```

## Créer un Statement

- L'interface `java.sql.Statement` représente une requête SQL
- Créer un Statement

```
stmt = connection.createStatement();
```

# Exécuter un Statement

- Requête sans modification de données :

```
ResultSet rs = stmt.executeQuery(sql);
```

- Requête avec modification de données :

```
int nbRows = stmt.executeUpdate(sql);
```

# Fermer un Statement

- **TOUJOURS FERMER LES STATEMENTS APRÈS UTILISATION :**

```
stmt.close();
```

# Le Développement de la couche Data Access Layer (DAL)

## Exploiter le résultat (**ResultSet**) d'un Statement

- **ResultSet :**  
données structurées de manière tabulaire
- Se déplacer dans les lignes
  - beforeFirst() ( Position par défaut )
  - first()
  - next()
  - previous()
  - last()
  - ...
- Récupérer les valeurs des colonnes de la ligne courante
  - Méthodes `getTypeColonne( colonne )`
    - `getString( colonne )`
    - `getInt( colonne )`
    - `getDate( colonne )`
    - ...
- Identifier une valeur NULL avec `wasNull()`

```
stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(sql);
Stagiaire stagiaire = null;
while(rs.next()){
    stagiaire = new Stagiaire();
    stagiaire.setNoStagiaire( rs.getInt("noStagiaire"));
    stagiaire.setPrenom( rs.getString("prenom"));
    stagiaire.setNom(rs.getString("nom"));
    stagiaire.setEmail( rs.getString("email"));
    stagiaire.setMotDePasse(rs.getString("motDePasse"));

    stagiaires.add(stagiaire);
}
```

## Se déplacer dans un ResultSet

- **createStatement( type, mode )**
  - Types de déplacement
    - ResultSet.TYPE\_FORWARD\_ONLY (*par défaut*)
    - ResultSet.TYPE\_SCROLL\_INSENSITIVE
    - ResultSet.TYPE\_SCROLL\_SENSITIVE
  - Mode d'ouverture
    - ResultSet.CONCUR\_READ\_ONLY (*par défaut*)
    - ResultSet.CONCUR\_UPDATEABLE

```
stmt = connection.createStatement(resultSet.TYPE_FORWARD_ONLY, resultSet.CONCUR_READ_ONLY);
```

# Le Développement de la couche Data Access Layer (DAL)

## La requête paramétrée

```
public void update(String prenom, String nom, int noStagiaire) throws SQLException{
    String sql = "update STAGIAIRES set prenom= " + prenom + ",nom=" + nom
                + " where noStagiaire=" + noStagiaire;

    Connection connection = JdbcTools.getInstance().getConnection();
    Statement stmt = connection.createStatement();

    stmt.executeUpdate(sql);
```

```
String sql = "update STAGIAIRES set prenom=?, nom=?,email=?,motDePasse=? where noStagiaire=?";
PreparedStatement stmt = connection.prepareStatement(sql);
```

# Le Développement de la couche Data Access Layer (DAL)

## Utiliser une requête paramétrée

```
String sql = "update STAGIAIRES set prenom=?, nom=? where noStagiaire=?";
```

- Créer un objet PreparedStatement

```
PreparedStatement stmt = connection.prepareStatement(sql);
```

- Fixer les valeurs des paramètres

```
stmt.setString(1, "Bob");
stmt.setString(2, "Leponge");
stmt.setInt(3, 10);
```

- Remarque : Utiliser la méthode `setNull()` pour initialiser un paramètre à Null

- Exécuter `stmt.executeUpdate();`

Le Développement de la couche Data Access Layer (DAL)  
Exécuter une requête SQL depuis Java

# Démonstration



# Le Développement de la couche Data Access Layer (DAL) Gestion d'une Papeterie - partie 2

TP



# Le Développement de la couche Data Access Layer (DAL)

## Appeler une procédure stockée

- Créer un objet **CallableStatement** :

```
String sql = "EXEC dbo.insertStagiaire @prenom =?, @nom =?, "
           + "@email=?, @noStagiaire=?";
CallableStatement callStmt = connection.prepareCall(sql);
```

- Renseigner les valeurs des Paramètres **IN** :

```
callStmt.setString(1, "Nordine");
callStmt.setString(2, "NATEUR");
```

- Définir les Paramètres **OUT** :

```
callStmt.registerOutParameter(4, java.sql.Types.INTEGER);
```

- **Exécuter** : `callStmt.execute();`

- Récupérer la valeur du Paramètre : `callStmt.getInt(4)`

# Le Développement de la couche Data Access Layer (DAL)

## Gérer les transactions

- Activer/Désactiver le mode auto-commit

```
connection.setAutoCommit(false);
```

- Gestion d'une transaction

- Début implicite

- Valider une transaction

```
connection.commit();
```

- Annuler les modifications

```
connection.rollback();
```

# Le Développement de la couche Data Access Layer (DAL)

## Externaliser la chaîne de connexion

- Par fichier Properties

- Format texte (extension .properties)

```
# Paramètres de connexion à SQLServer
driverdb=com.microsoft.sqlserver.jdbc.SQLServerDriver
urldb=jdbc:sqlserver://127.0.0.1;dbname=EVAL_DB;
userdb=sa
passworddb=Pa$$w0rd
```

- Format XML ( DTD : <http://java.sun.com/dtd/properties.dtd>)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>Paramètres de connexion à SQLServer</comment>
    <entry key="driverdb">com.microsoft.sqlserver.jdbc.SQLServerDriver</entry>
    <entry key="urldb">jdbc:sqlserver://127.0.0.1;dbname=EVAL_DB</entry>
    <entry key="userdb">sa</entry>
    <entry key="passworddb">Pa$$w0rd</entry>
</properties>
```

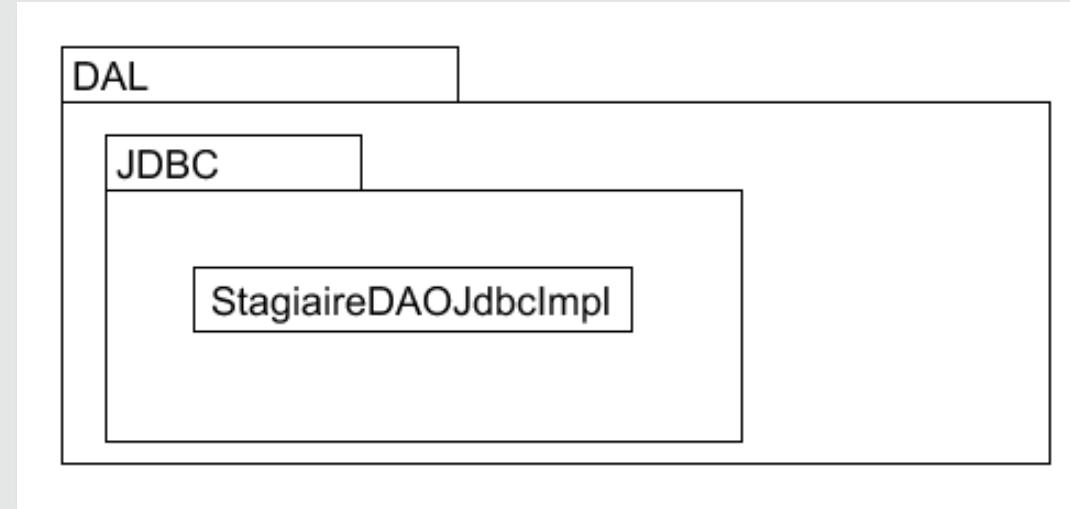
Le Développement de la couche Data Access Layer (DAL)  
Externaliser la chaîne de connexion

# Démonstration



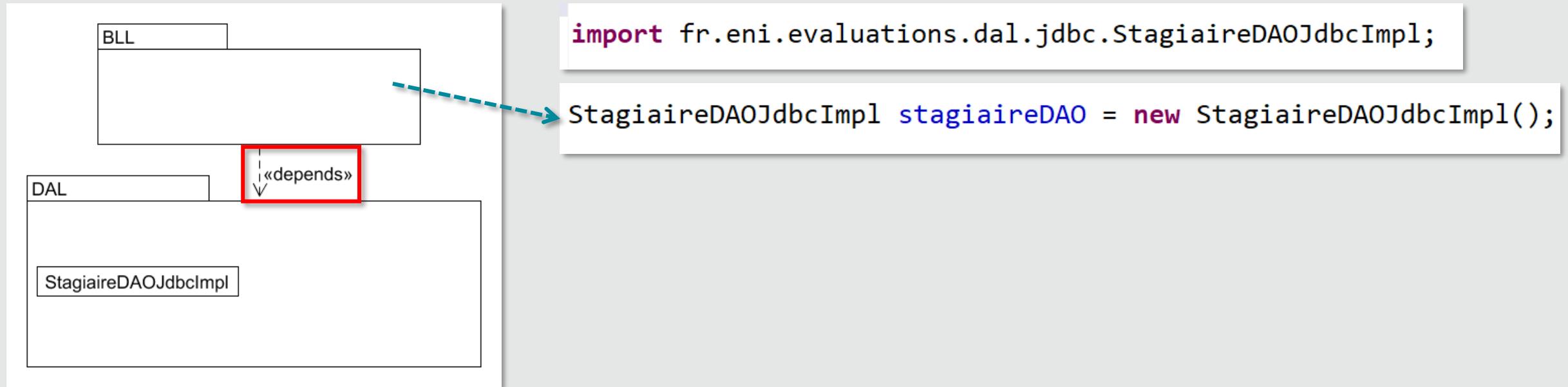
# Le Développement de la couche Data Access Layer (DAL)

## Définir le Data Access Object (DAO)



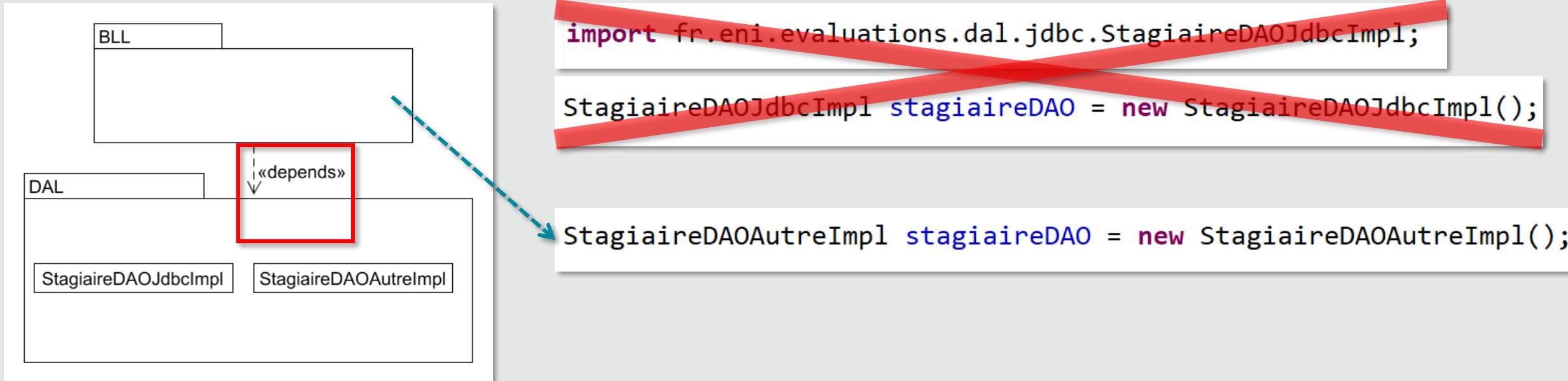
# Le Développement de la couche Data Access Layer (DAL)

## Utiliser la DAL depuis la BLL



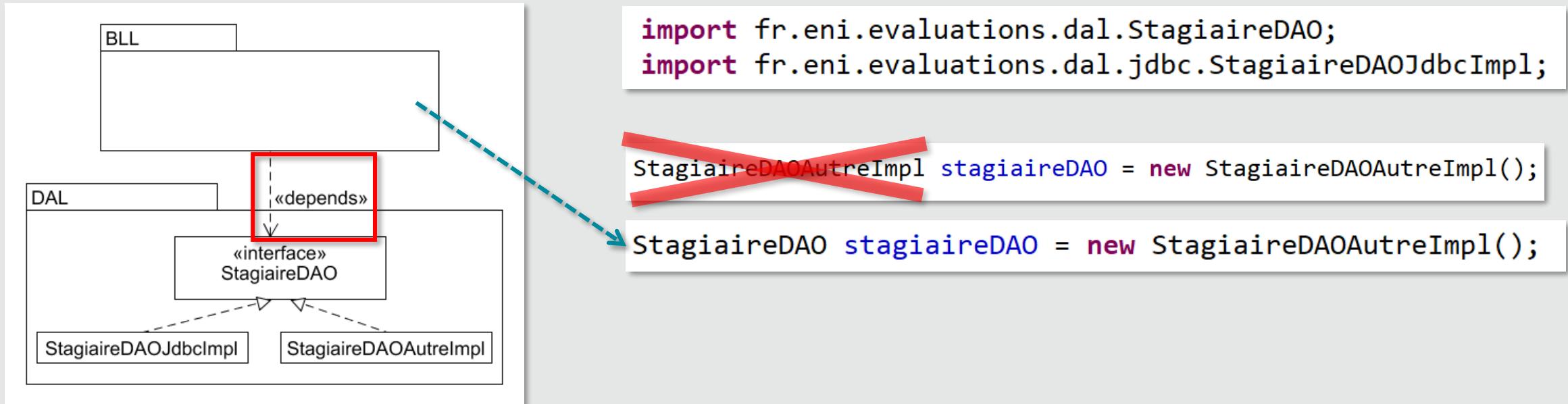
# Le Développement de la couche Data Access Layer (DAL)

## Utiliser la DAL depuis la BLL



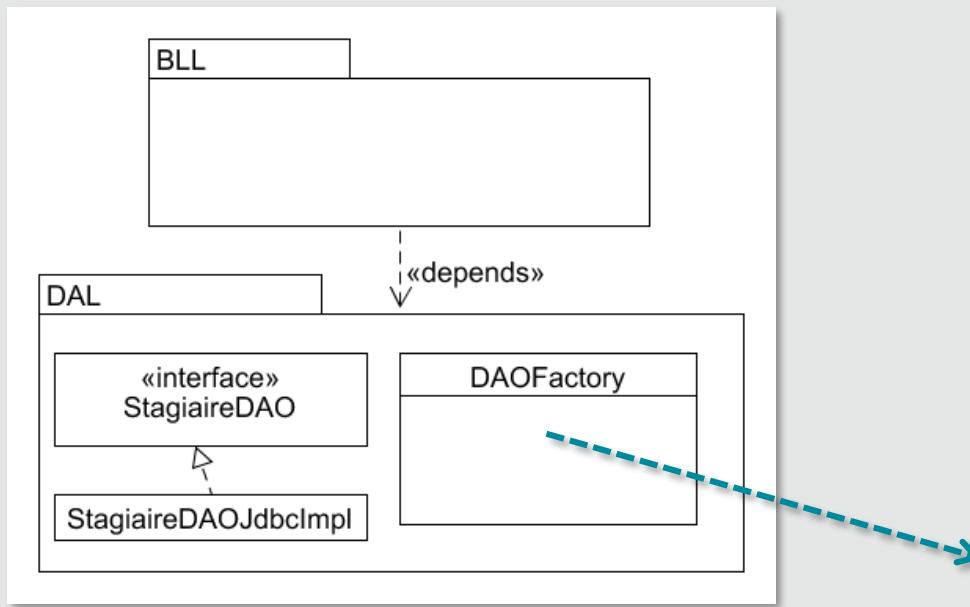
# Le Développement de la couche Data Access Layer (DAL)

## Utiliser la DAL depuis la BLL



# Le Développement de la couche Data Access Layer (DAL)

## Utiliser une Factory



```
StagiaireDAO stagiaireDAO = new StagiaireDAOJdbcImpl();
```

```
StagiaireDAO stagiaireDAO = DAOFactory.getStagiaireDAO();
```

```
public class DAOFactory {
```

```
    public static StagiaireDAO getStagiaireDAO() {
```

```
        StagiaireDAO stagiaireDAO= new StagiaireDAOJdbcImpl();
```

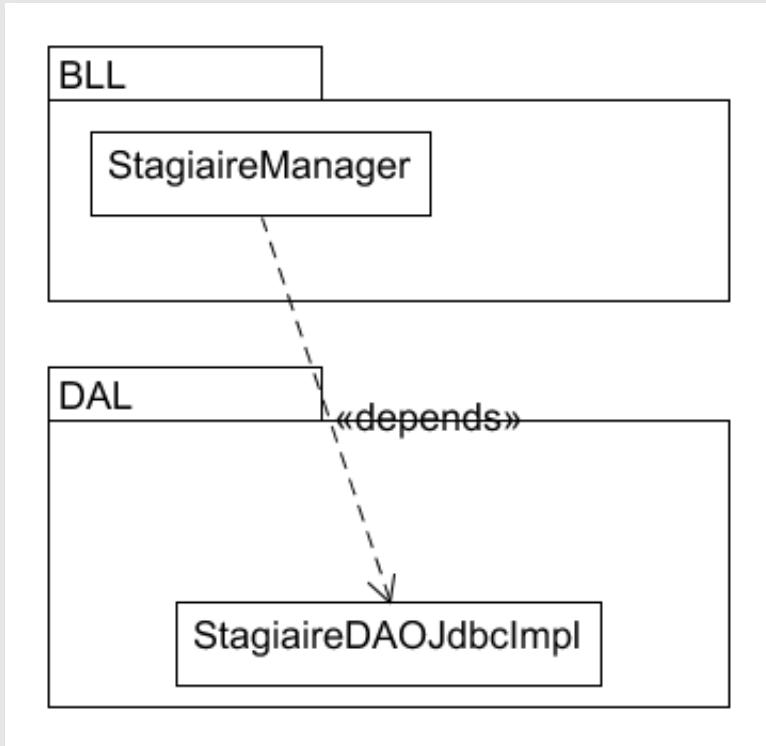
```
        return stagiaireDAO;
```

```
    }
```

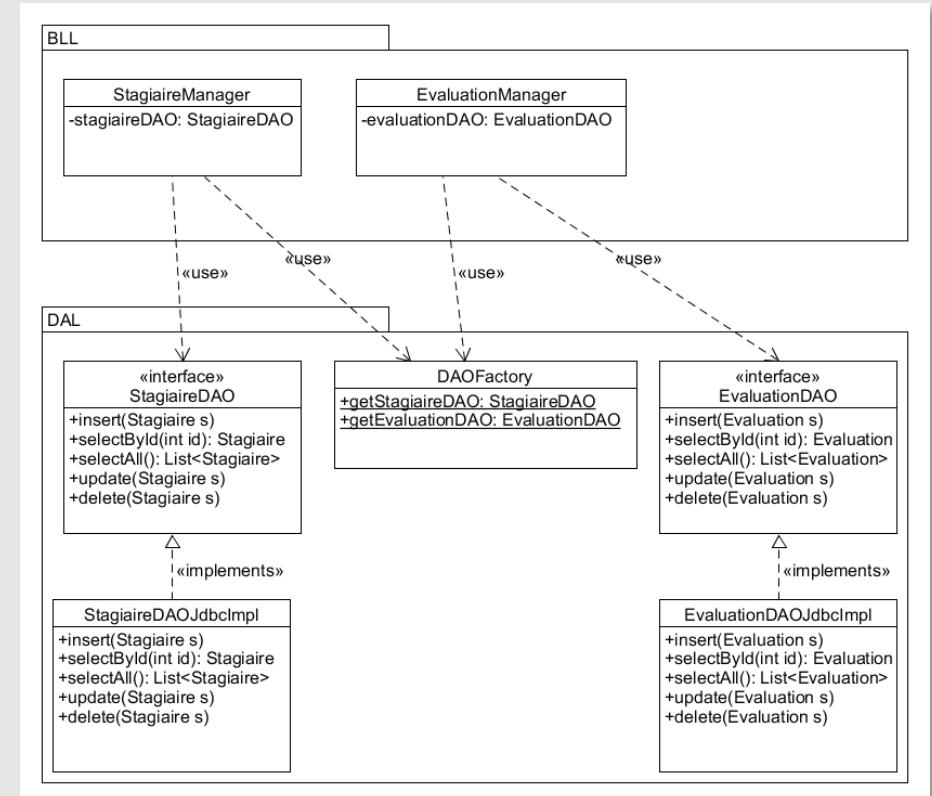
```
}
```

# Le Développement de la couche Data Access Layer (DAL) Utiliser la DAL depuis la BLL

AVANT

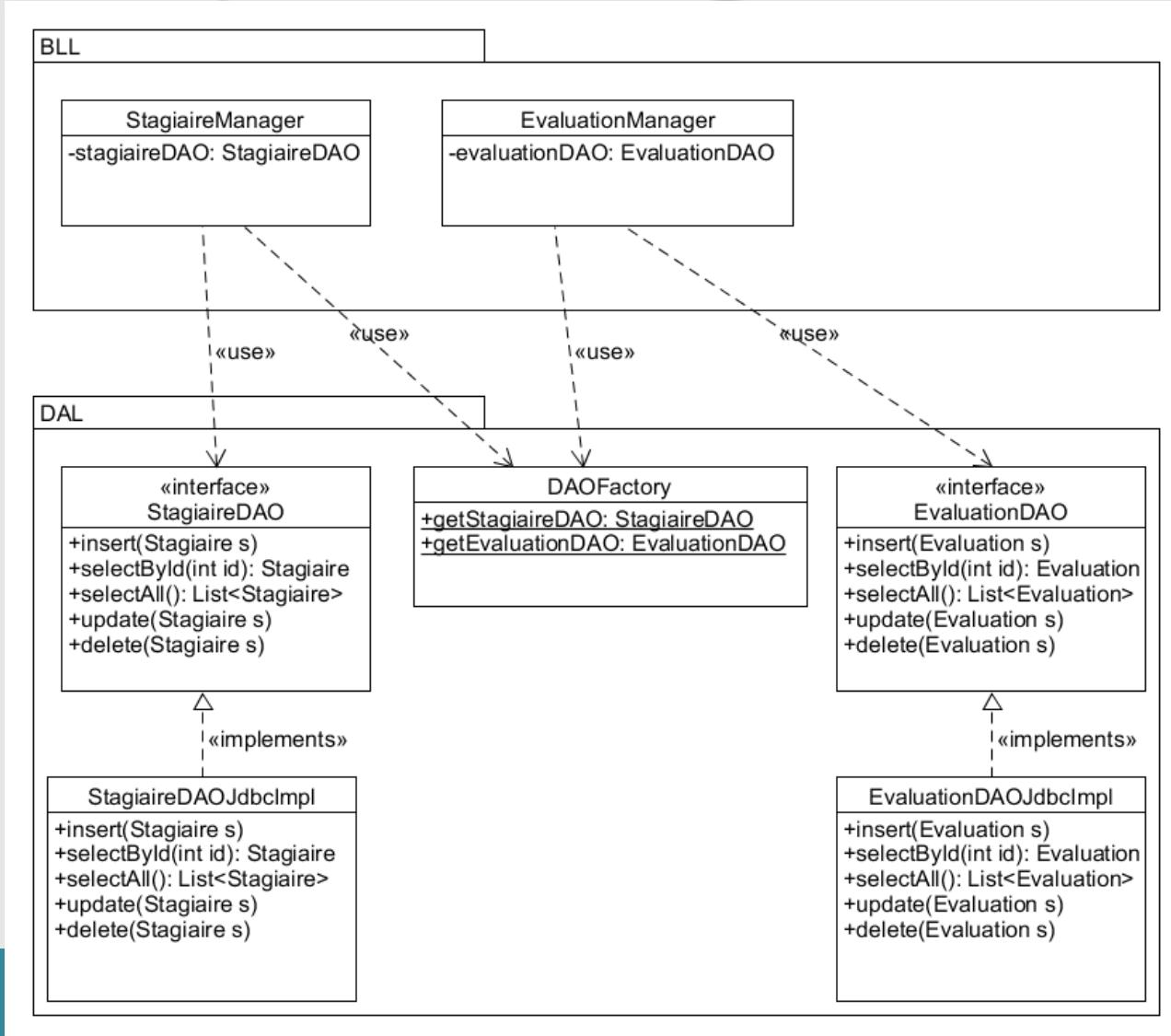


APRÈS



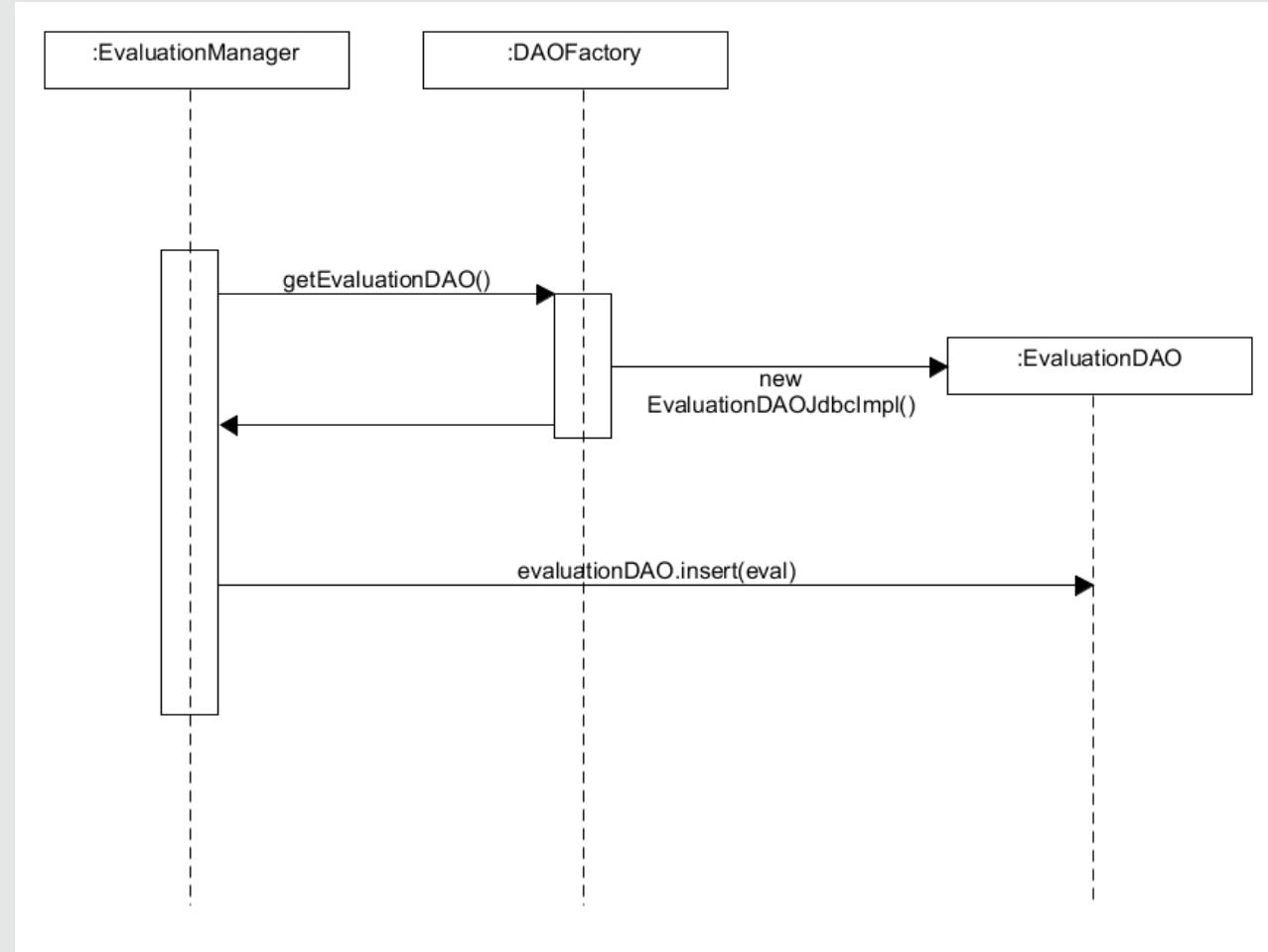
# Le Développement de la couche Data Access Layer (DAL)

## Le modèle statique du *Design Pattern DAO*



# Le Développement de la couche Data Access Layer (DAL)

## Le modèle dynamique du *Design Pattern DAO*



Le Développement de la couche Data Access Layer (DAL)  
Le Design Pattern DAO

# Démonstration



# Le Développement de la couche Data Access Layer (DAL) Gestion d'une papeterie - partie 3

TP



# Le Développement en Couches ... avec Java SE

**Module 05 – Le Développement de la Couche  
Business Logic Layer (BLL)**

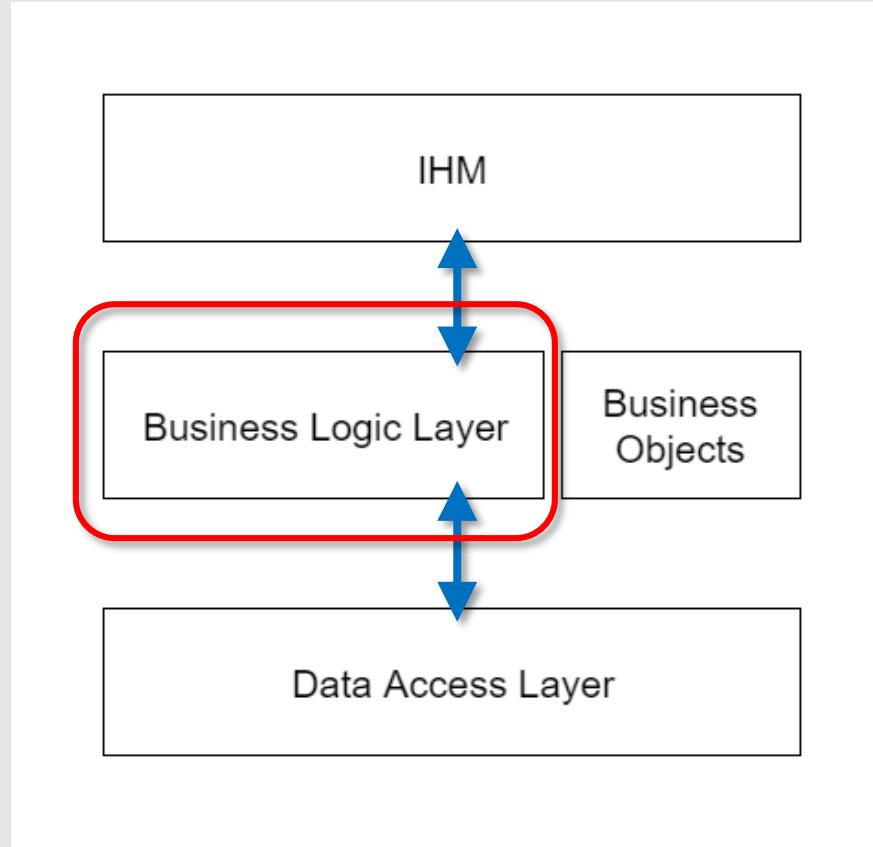


# Objectifs

- Situer la couche BLL
- Responsabilités de la couche BLL
- Comprendre l'organisation de la couche BLL
- Savoir implémenter la couche BLL

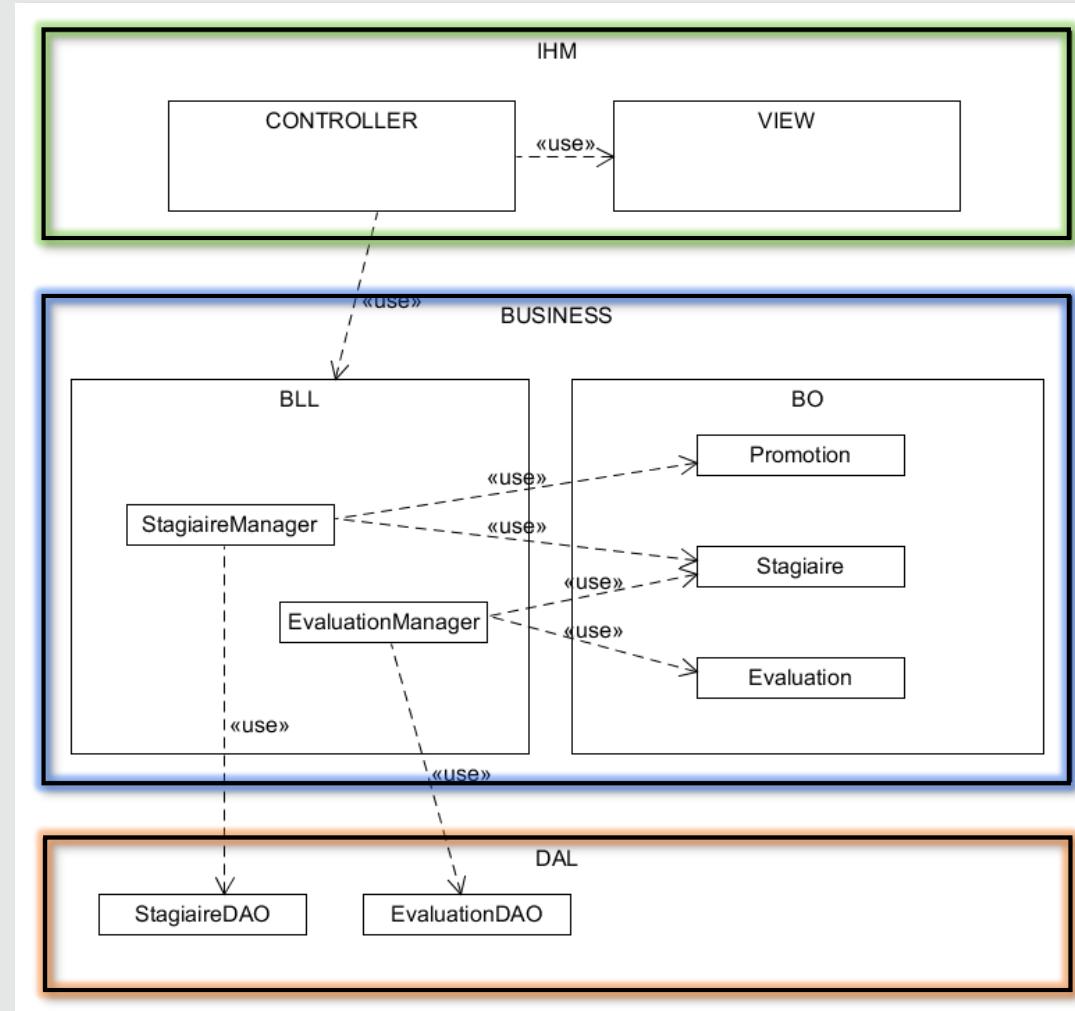
Le Développement de la couche Business Logic Layer (BLL)

# Situer la couche Business Logic Layer (BLL)



# Le Développement de la couche Business Logic Layer (BLL)

## La composition de la couche BLL



## Le Développement de la couche Business Logic Layer (BLL)

# L'utilisation de la couche BLL par instantiation directe

```
EvaluationManager evalMger = new EvaluationManager();
float moyenne = evalMger.calculerMoyenne(stagiaire);
```

- Avantage

- Solution naturelle et simple

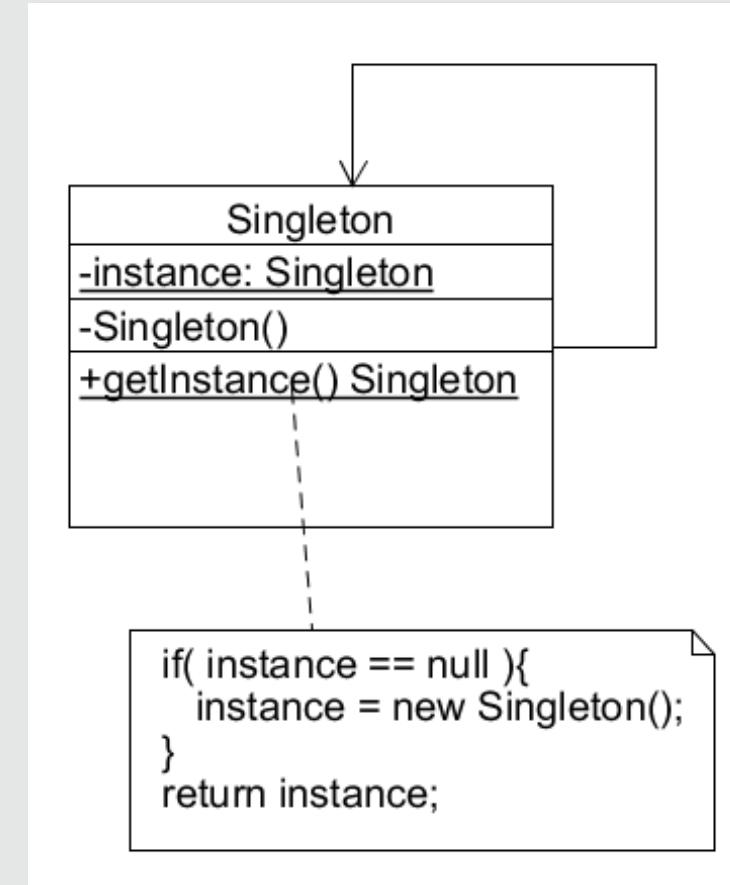
- Inconvénients

- Performances et espace mémoire à maîtriser
  - Plusieurs instances de EvaluationManager sont créées
  - Risque de duplication de données

# Le Développement de la couche Business Logic Layer (BLL)

## Le *Design Pattern Singleton*

- Objectif : 1 seule instance
- Attribut instance statique
- Méthode getInstance()
  - Gère la création de l'instance
  - Méthode statique
- Constructeur privé



# Le Développement de la couche Business Logic Layer (BLL)

## Appliquer le *Design Pattern Singleton*

```
public class EvaluationManager {  
  
    private static EvaluationManager instance;  
  
    public static EvaluationManager getInstance(){  
        if(instance == null){  
            instance = new EvaluationManager();  
        }  
        return instance;  
    }  
  
    private EvaluationManager(){  
    }  
  
    public float calculerMoyenne( List<Passage> passages) throws BLLException {  
  
        if(passages == null || passages.size()==0){  
            throw new BLLException("Aucune évaluation");  
        }  
  
        float total = 0f;  
        for(Passage passage:passages){  
            total+=passage.getNote();  
        }  
        return total/passages.size();  
    }  
}
```

# Le Développement de la couche Business Logic Layer (BLL) Gestion d'une papeterie - partie 4

TP



# Le Développement en Couches ... avec Java SE

**Module 06 – Développer la Couche IHM (GUI)  
avec Swing**

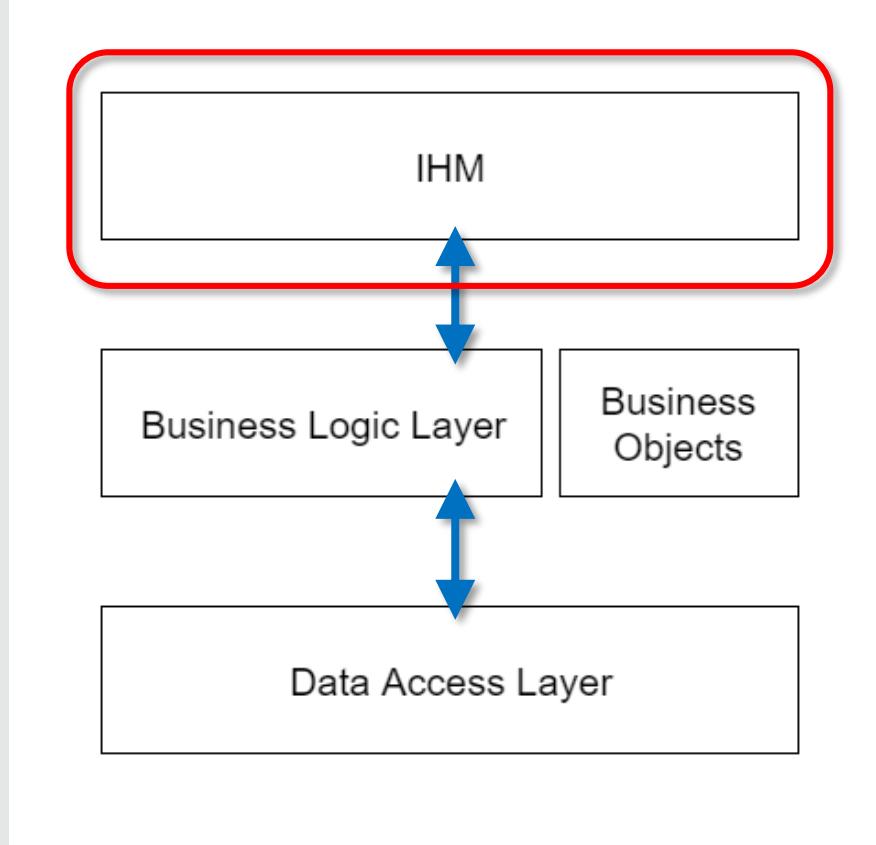


## Objectifs

- Implémenter la couche IHM avec l'API Swing
- Comprendre les principes de la programmation événementielle
- Comprendre le Design Pattern Observer

Développer la couche IHM avec Swing

# Situer la couche Interface Humain-Machine (IHM)



Développer la couche IHM avec Swing

# L'API Swing

- Composants graphiques Swing (Boutons, Champs, Tables, Panneaux...)
- Gestion du Look and Feel
- Autre
  - Accessibilité
  - API 2D
  - Internationalisation

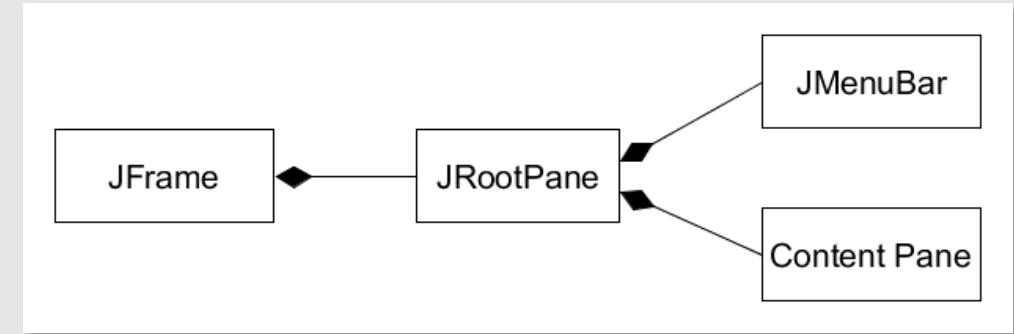
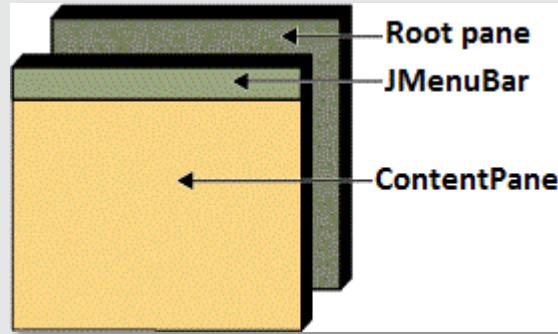
javax.swing

# Développer la couche IHM avec Swing

## La structure d'un écran



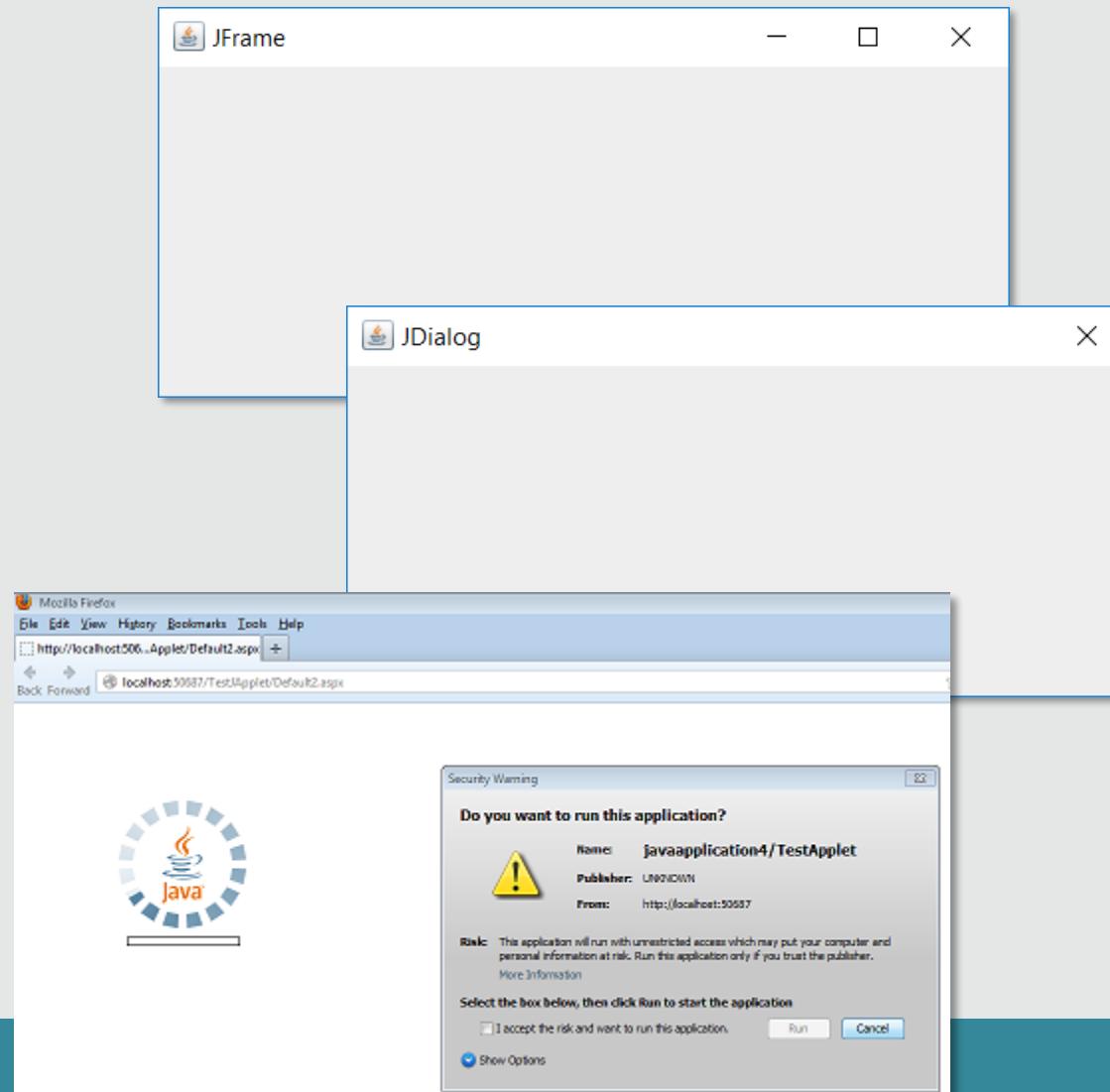
Conteneur de haut niveau



# Développer la couche IHM avec Swing

## Conteneurs de plus haut niveau

- JFrame
- JDialog
- JApplet



# Développer la couche IHM avec Swing

## Lancer une application Swing

```
public class EcranPrincipal extends JFrame

public class EvaluationApp {

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                EcranPrincipal frame = new EcranPrincipal();
            }
        });
    }
}
```

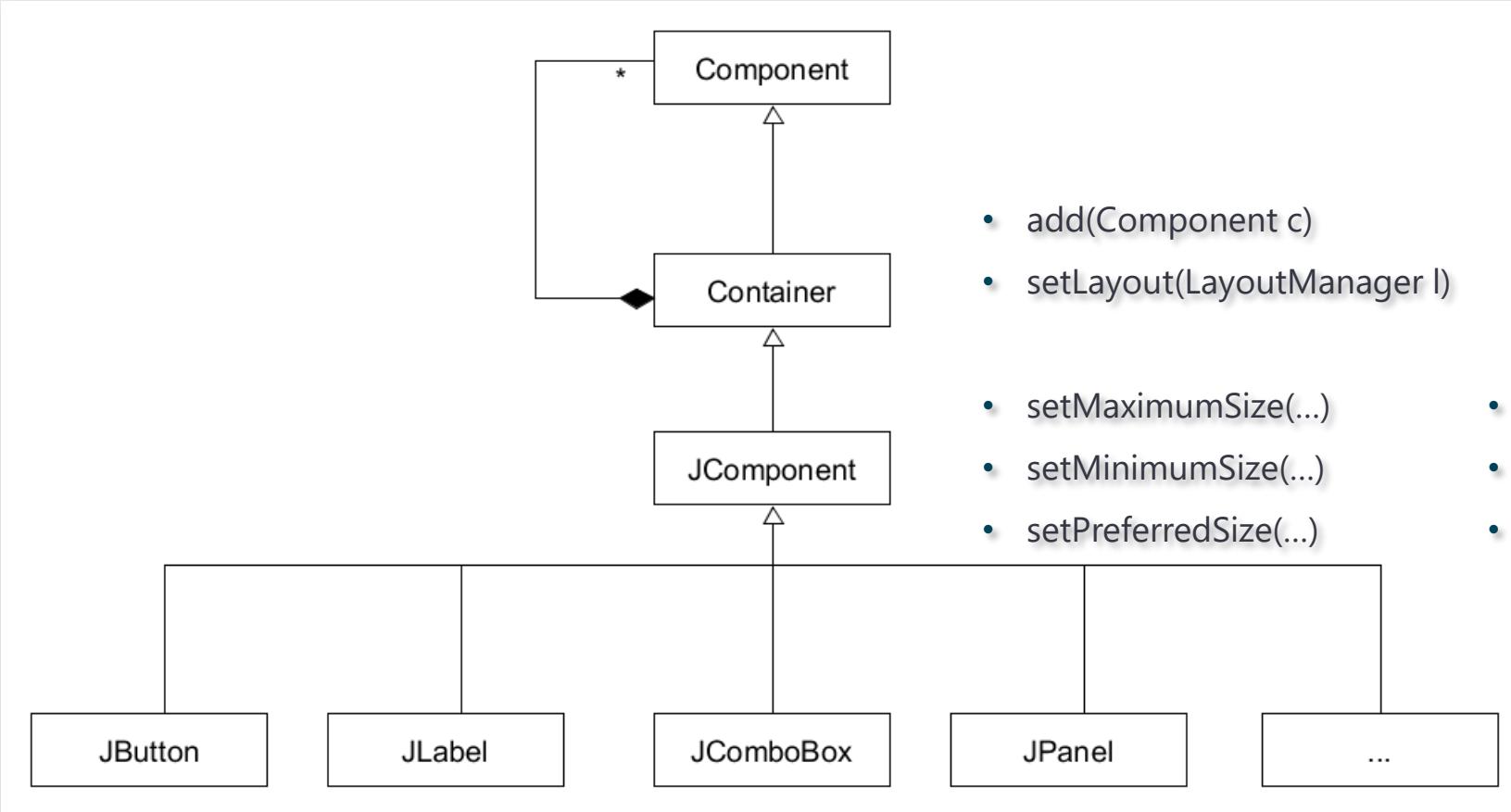
Développer la couche IHM avec Swing  
Coder une fenêtre de type JFrame

# Démonstration



# Développer la couche IHM avec Swing

## Les composants graphiques



# Placer les composants sur la JFrame

- Affecter un JPanel à la JFrame via le contentPane

```
public EcranStagiaire() {  
    super("Ajout Stagiaire");  
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
    setSize(500, 300);  
    setResizable(false);  
    setLocationRelativeTo(null);  
    this.setContentPane(getPanelPrincipal());  
}
```

Développer la couche IHM avec Swing

# Ajouter les Composants Graphiques

```
private void initIHM(){
    JPanel panel = new JPanel();
    panel.setOpaque(true);

    panel.add(getTxtNom());

    //Ajouter le panel à la JFrame
    this.setContentPane(panel);
}

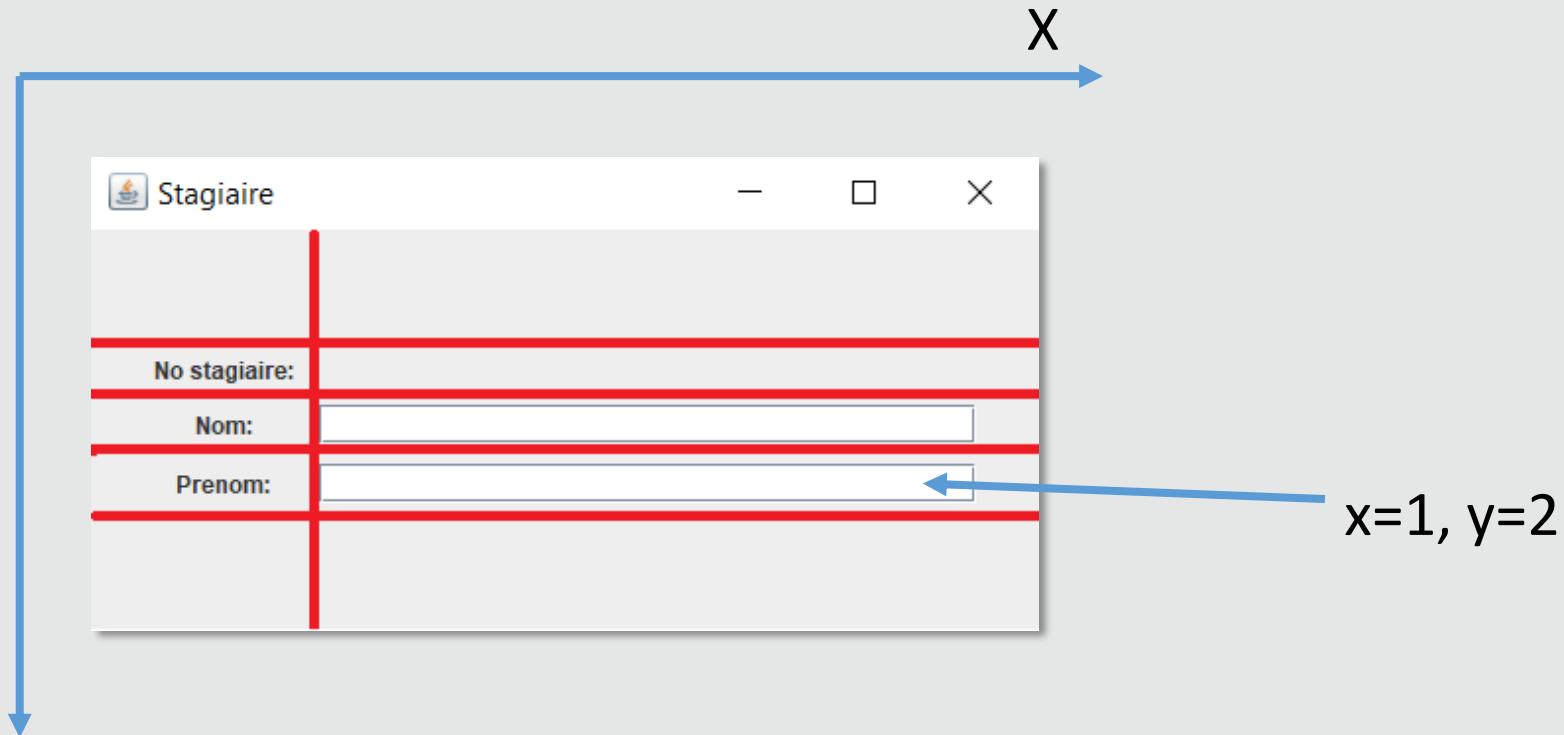
public JTextField getTxtNom() {
    if(txtNom==null){
        txtNom = new JTextField(30);
    }
    return txtNom;
}
```

# Utiliser les Layouts

- Le Layout permet de gérer la disposition des éléments sur le conteneur
- Chaque conteneur a un Layout associé par défaut
  - Le Layout par défaut du **JFrame** est le **BorderLayout**
  - Le Layout par défaut du **JPanel** est le **FlowLayout**
- Gestion du Layout
  - `monContainer.setLayout(LayoutManager mgr)`
  - `monContainer.add(Component c)`
- Différents layouts
  - <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

Développer la couche IHM avec Swing

# Le GridBagLayout



# Développer la couche IHM avec Swing

## Le GridBagLayout

```
public JPanel getPanelPrincipal(){  
    JPanel panel = new JPanel();  
    panel.setOpaque(true); //Recommandé par Oracle  
    panel.setLayout(new GridBagLayout());  
    GridBagConstraints gbc = new GridBagConstraints();  
  
    //Ligne 1  
    gbc.gridx = 0;  
    gbc.gridy = 0;  
    gbc.insets = new Insets(5,5,5,5);  
    panel.add(getLblPrenom(), gbc);  
    gbc.gridx = 1;  
    panel.add(getTxtPrenom(), gbc);  
    //Ligne 2  
    gbc.gridx = 1;  
    gbc.gridy = 1;  
    panel.add(getLblNom(), gbc);  
    gbc.gridx = 1;  
    panel.add(getTxtNom(), gbc);  
    //...  
    return panel;  
}
```

```
public JLabel getLblPrenom() {  
    if(lblPrenom==null){  
        lblPrenom = new JLabel("Prénom: ");  
    }  
  
    return lblPrenom;  
}
```

Développer la couche IHM avec Swing  
Utiliser un GridBagLayout

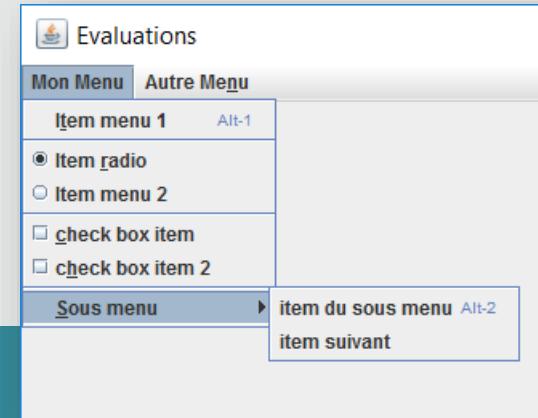
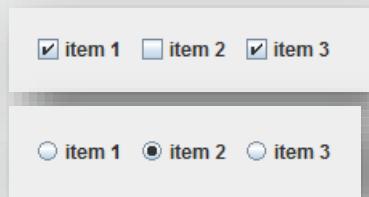
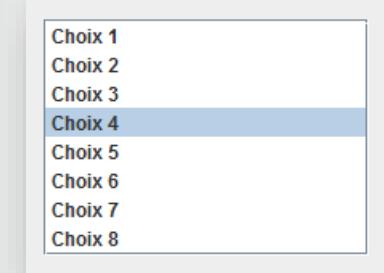
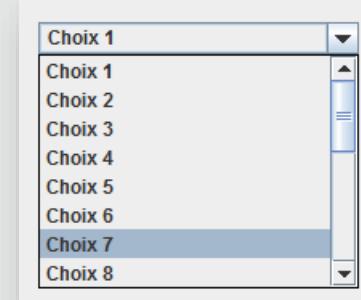
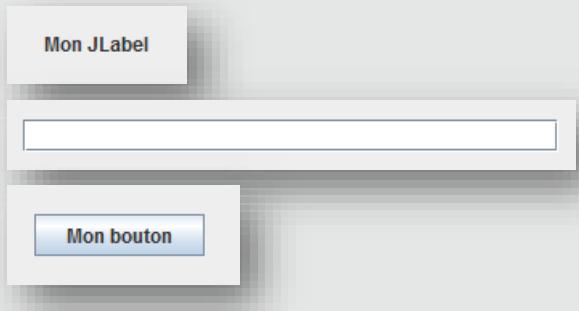
# Démonstration



# Développer la couche IHM avec Swing

## Les composants graphiques principaux

- JLabel
- JTextField
- JButton
- Jlist
- JComboBox
- ButtonGroup
  - JCheckBox
  - JRadioButton
- JToolBar, JMenuBar, JMenu



# Développer la couche IHM avec Swing

## Appliquer un look and feel

- Par programmation

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

```
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
```

- En ligne de commande

```
java -Dswing.defaultlaf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel MyApp
```

- Par fichier de configuration

- Swing.properties
- Swing.defaultlaf=com.sun.java.swing.plaf.windows.WindowsLookAndFeel

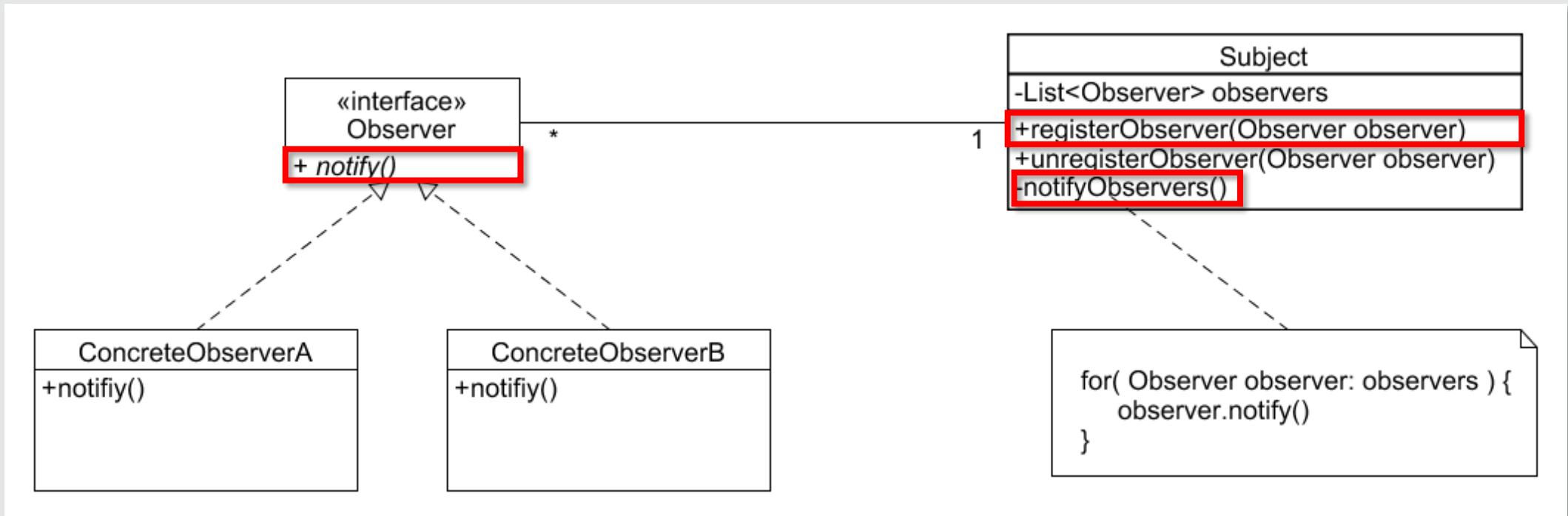
# La programmation événementielle

- Caractérisée par l'exécution de code lors de l'apparition d'événements
- Un événement peut être un clic sur un bouton, le déplacement de la souris au dessus d'un composant, le changement de la valeur d'un champ de saisie, etc.
- Bien adaptée au développement des applications graphiques

# Développer la couche IHM avec Swing

## La programmation événementielle

- Pattern Observer :



Développer la couche IHM avec Swing

# Programmation événementielle dans Swing

- Inscription d'un Listener : addTypeEvenementListener
- Exemples :

```
btnAjout.addActionListener(new AjoutStagiaireListener()));
```

```
zoneTexte.addFocusListener(...)
```

```
zoneTexte.addMouseListener(
```

## Développer la couche IHM avec Swing

# Programmation événementielle dans Swing

- Définir un Listener avec une classe interne

```
private class AjoutStagiaireListener implements ActionListener {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(EcranStagiaire.this, "Click sur Ajout Stagiaire");  
    }  
}
```

- Définir un Listener avec une classe anonyme

```
btnAjout.addActionListener(new ActionListener(){  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(EcranStagiaire.this, "Click sur Ajout Stagiaire");  
    }  
});
```

# Développer la couche IHM avec Swing

# Gestion d'une papeterie - partie 5

TP



# Développer la couche IHM avec Swing

# Gestion d'une papeterie - partie 6

TP



# Développer la couche IHM avec Swing WindowBuilder

## Démonstration



# Le Développement en Couches ... avec Java SE

**Module 07 – L'architecture Modèle Vue Contrôleur (MVC)**

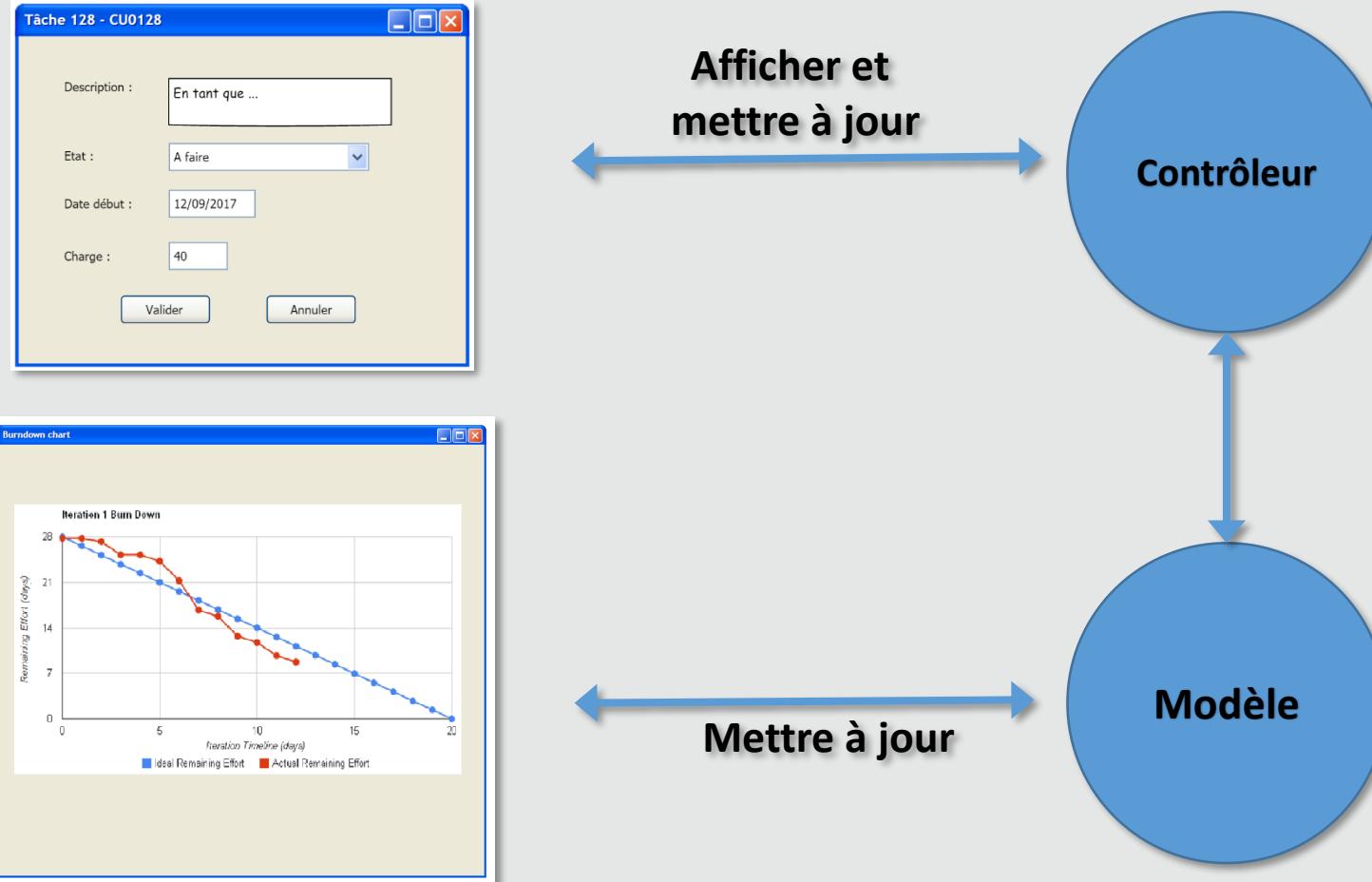


# Objectifs

- Comprendre et implémenter l'architecture Modèle Vue Contrôleur (MVC)
- Appliquer le Design Pattern Observer

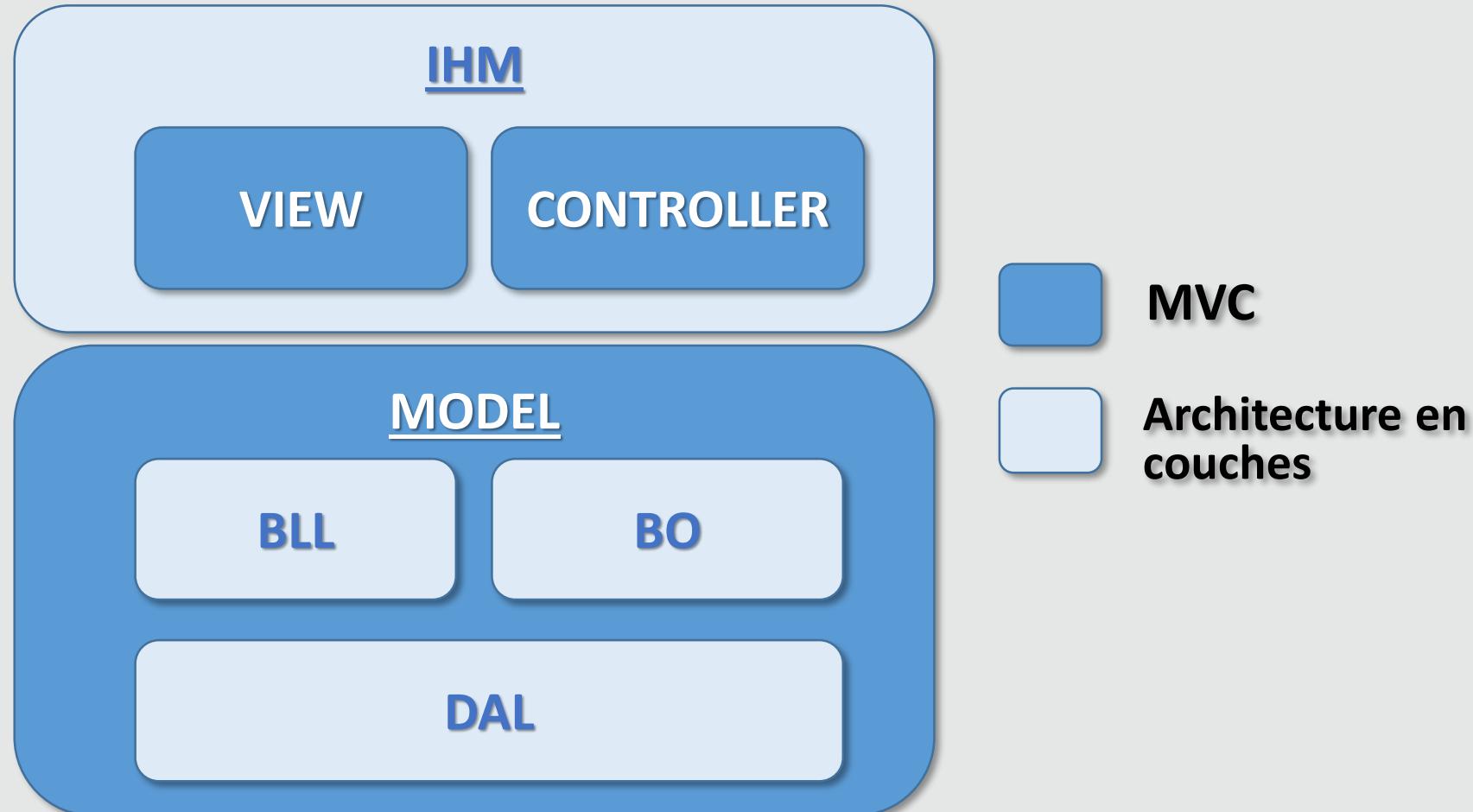
# L'Architecture Modèle Vue Contrôleur (MVC)

## Modèle MVC : les origines



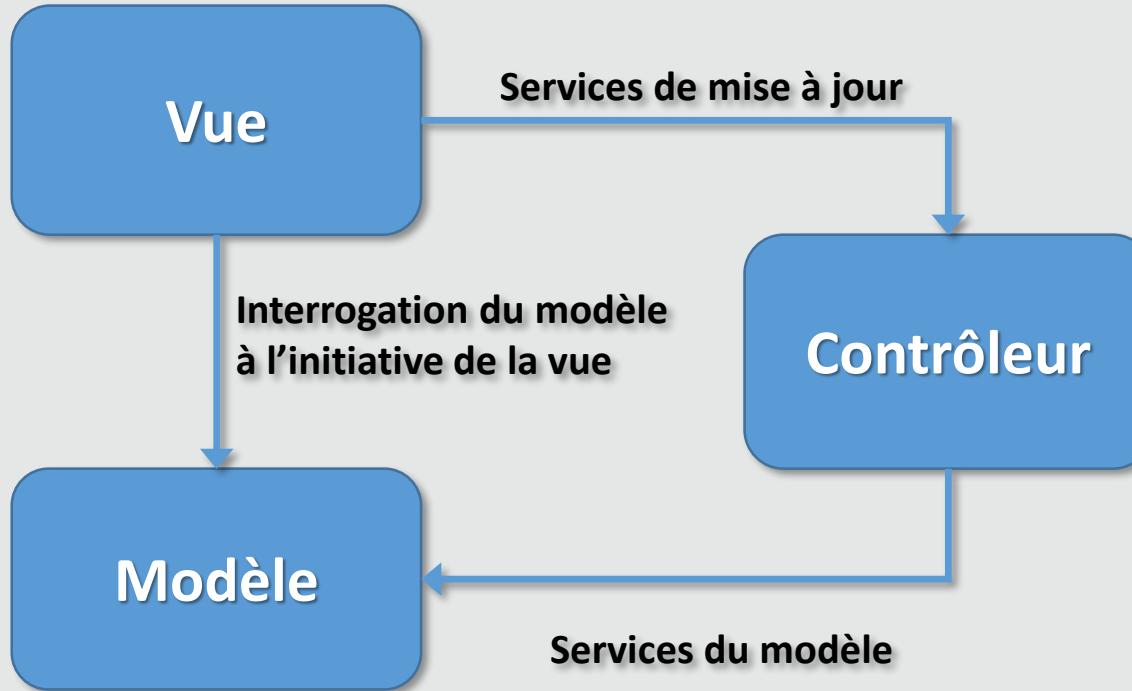
# L'Architecture Modèle Vue Contrôleur (MVC)

## Faire correspondre MVC et couches



# L'Architecture Modèle Vue Contrôleur (MVC)

## Mise à jour de la vue en méthode pull

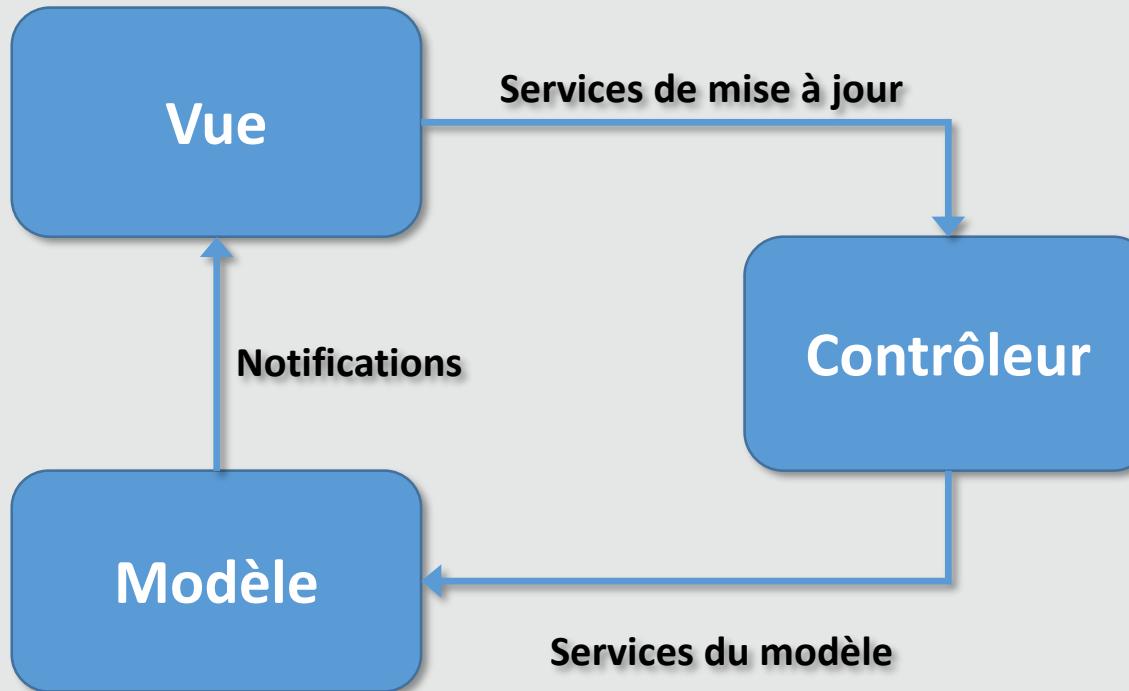


### Inconvénients :

- Risque d'affichage de données obsolètes
- Solution coûteuse en nombre de requêtes

# L'Architecture Modèle Vue Contrôleur (MVC)

## Mise à jour de la vue en méthode push



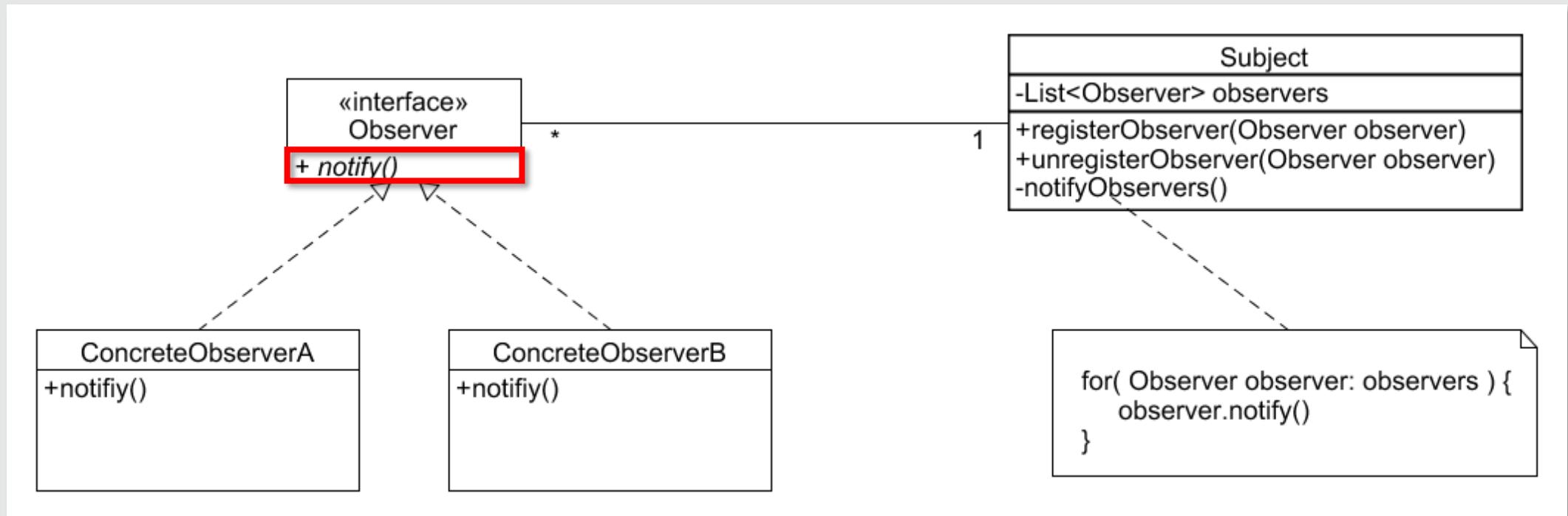
### Avantages :

- Actualisation des données affichées en temps réel
- Mécanisme de notification unique pour des vues différentes
- Découpler la vue et le modèle

# L'Architecture Modèle Vue Contrôleur (MVC)

## Implémenter le push avec le pattern Observer

- Pattern Observer :



# L'Architecture Modèle Vue Contrôleur (MVC)

# Gestion d'une papeterie – partie 7

TP

# Le Développement en Couches ... avec Java SE

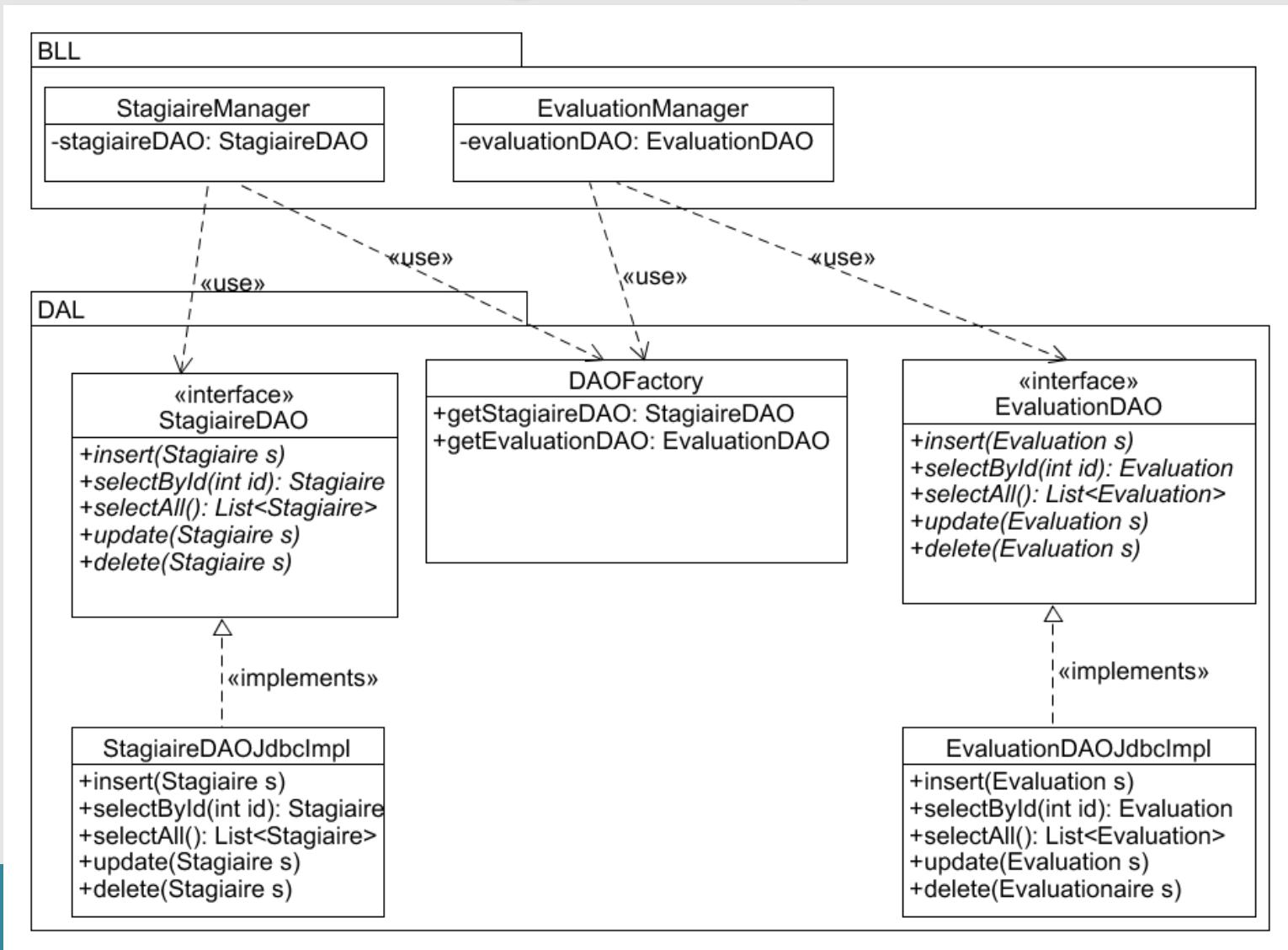
**Module 08 – Notions Avancées**



# Objectifs

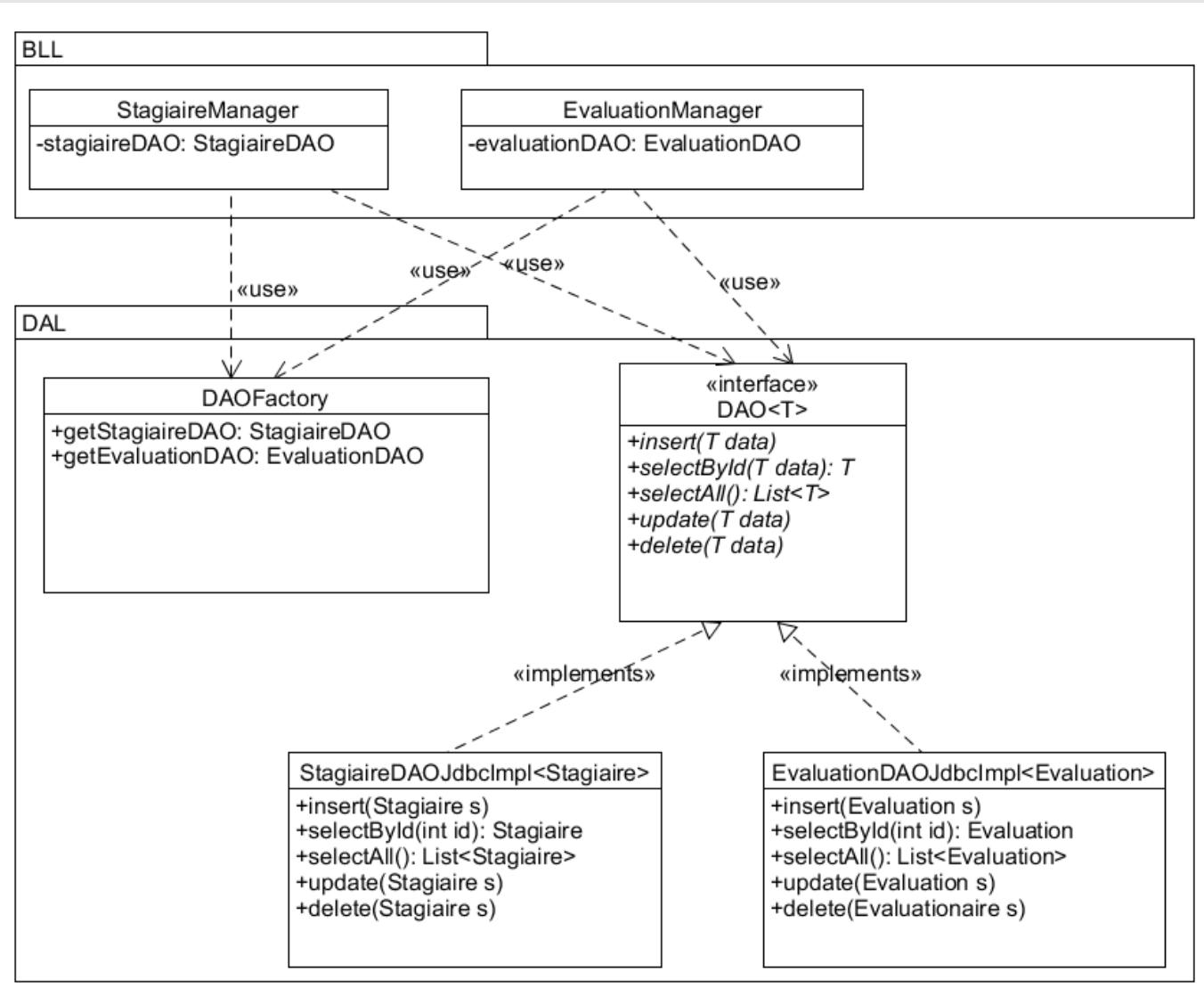
- Appliquer la généricité au pattern DAO
- Introduire la notion de N-TIERS

# Le pattern DAO non générique



# Le pattern DAO générique

Version  
avec générativité



Notions Avancées

# Pattern DAO Générique

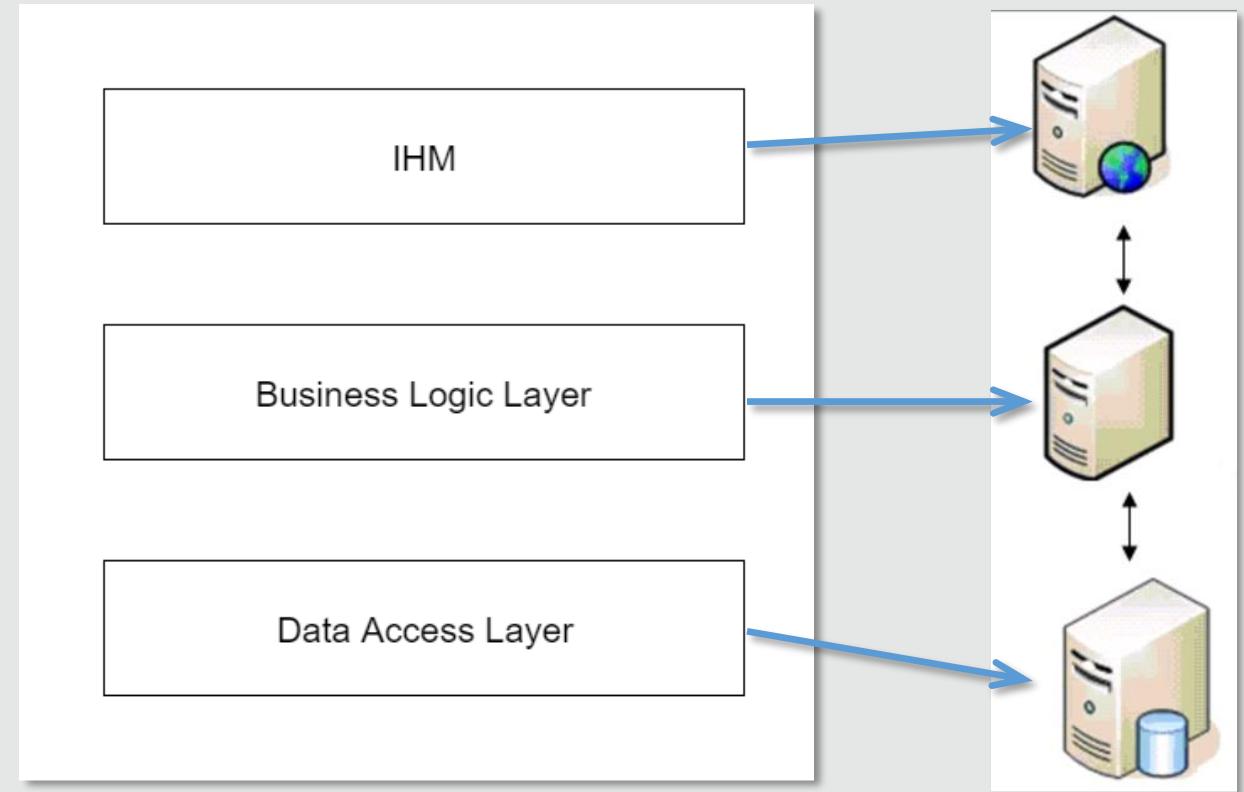
## Démonstration



# Notions Avancées

## Le N-TIERS

- Le N-TIERS désigne la distribution des couches sur l'architecture physique
- COUCHE = LOGICIEL
- TIERS = NŒUD PHYSIQUE



Notions Avancées

# Gestion d'une papeterie – partie 8

TP



# Le Développement en Couches ... avec Java SE

**La formation est à présent terminée,  
merci encore pour votre attention.**

