



Chapitre 10

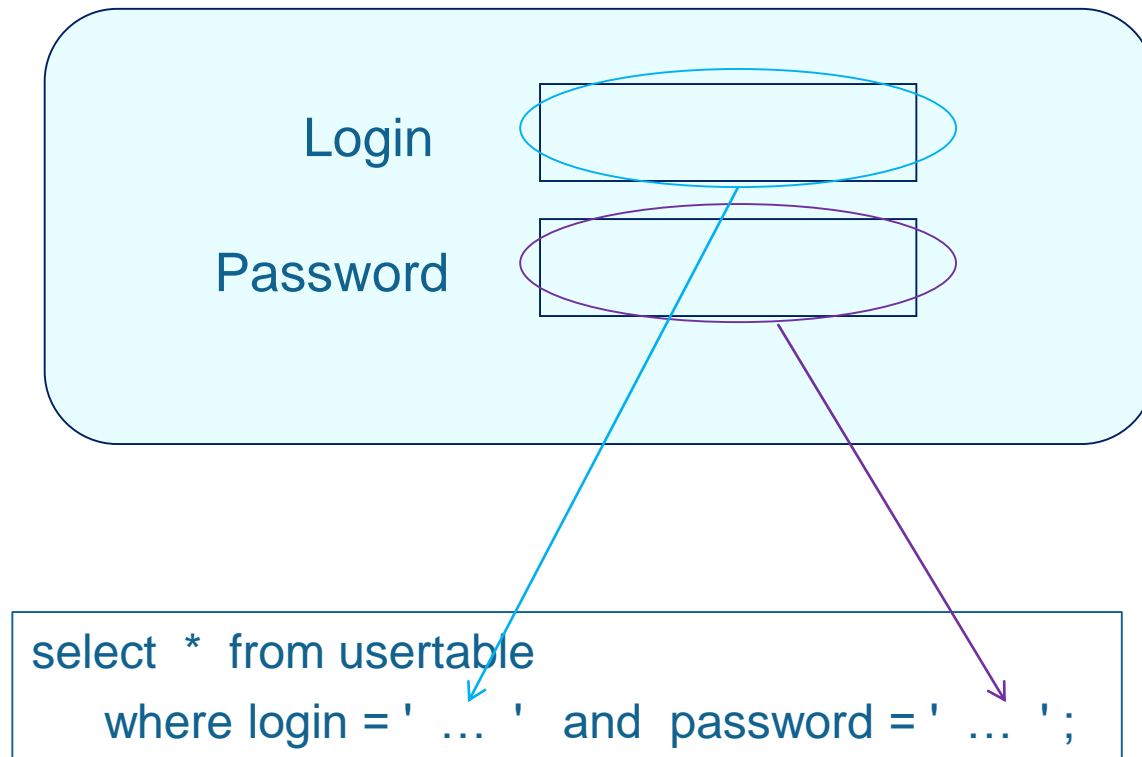
Accès aux bases de données

Comment établir une connexion entre un programme Java et une base de données et exécuter des instructions SQL à partir d'un code Java

1. Injection SQL

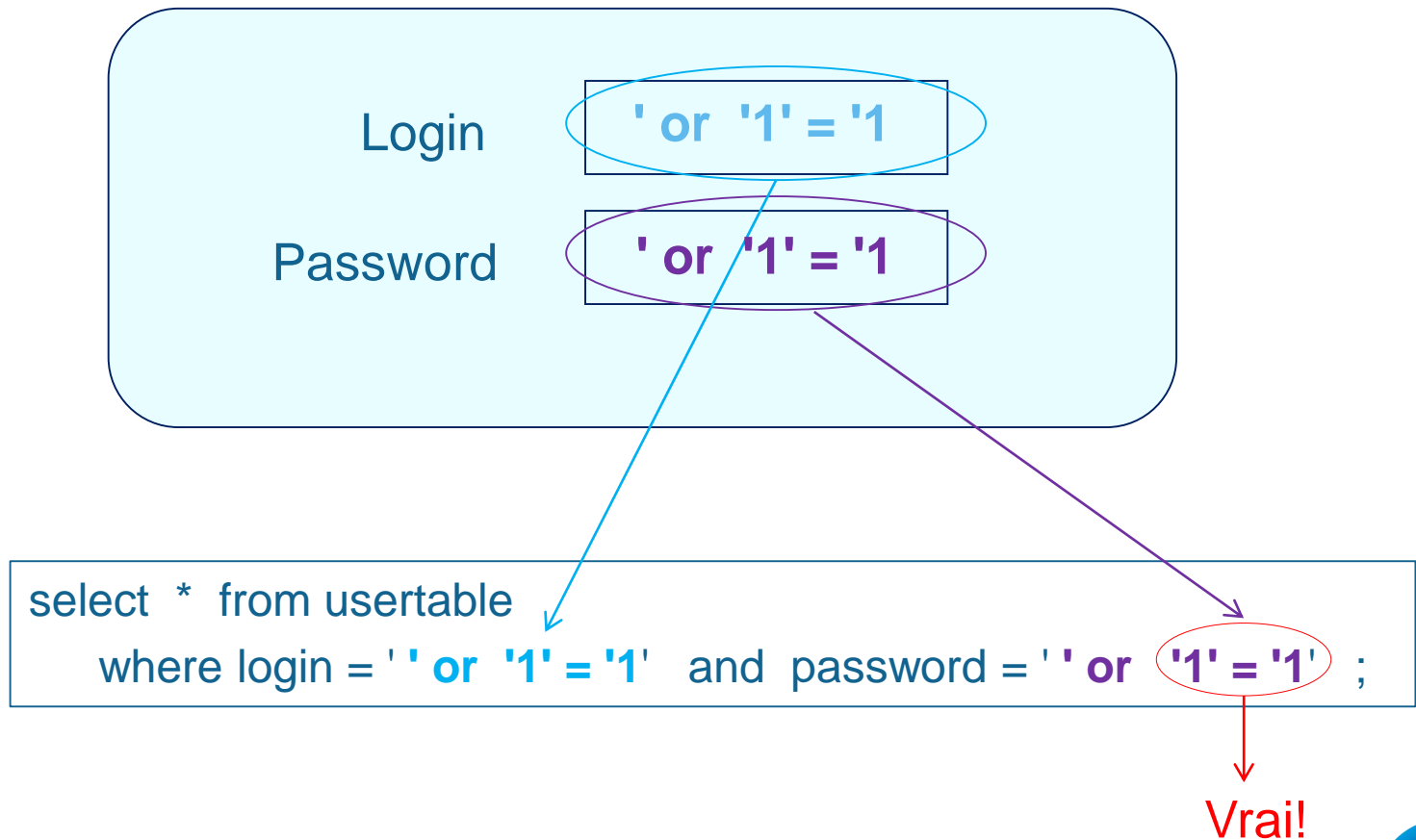
Injection SQL

- Pour "bypasser" l'étape d'authentification



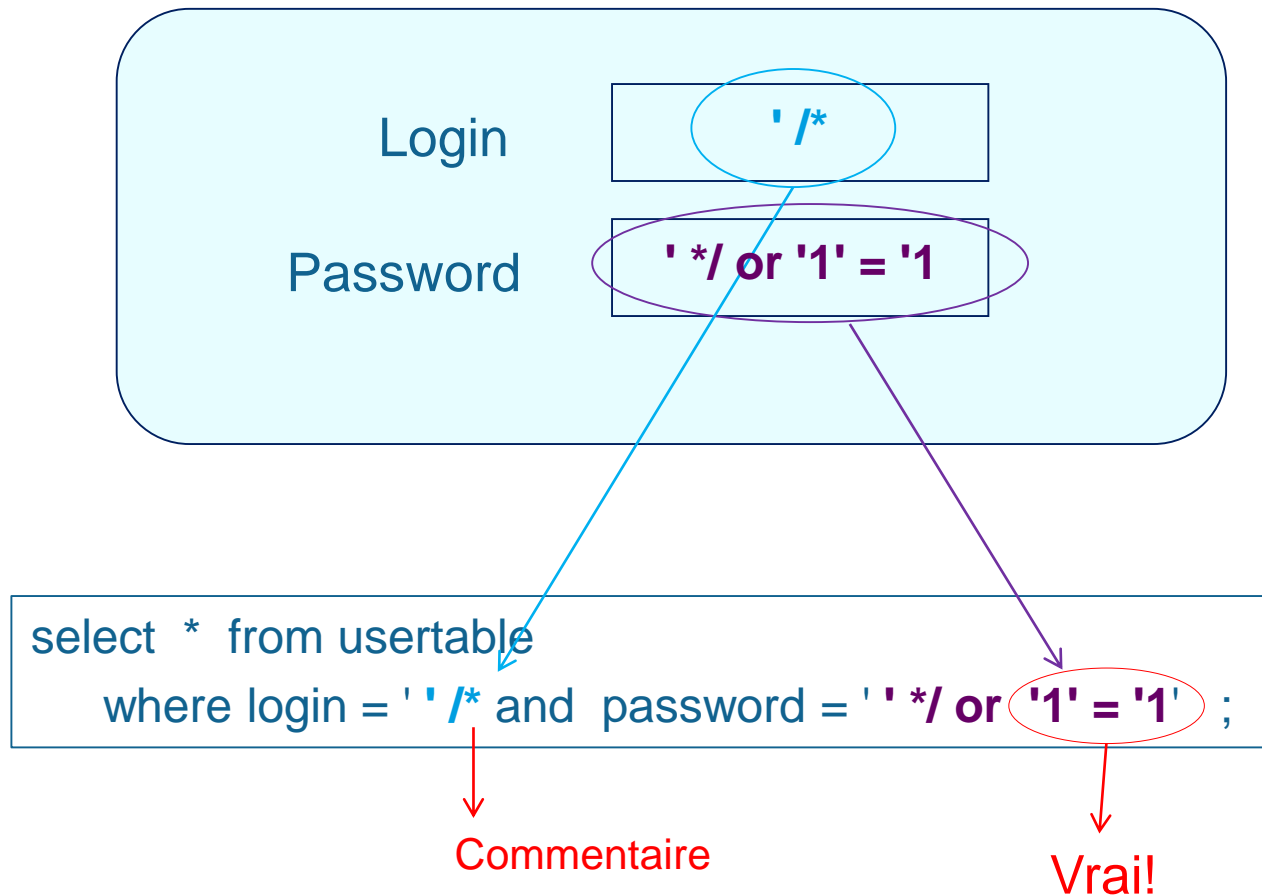
Injection SQL

- Exemple



Injection SQL

- En MySQL



Injection SQL

- Pour injecter des données dans une base de données
 - Exemple

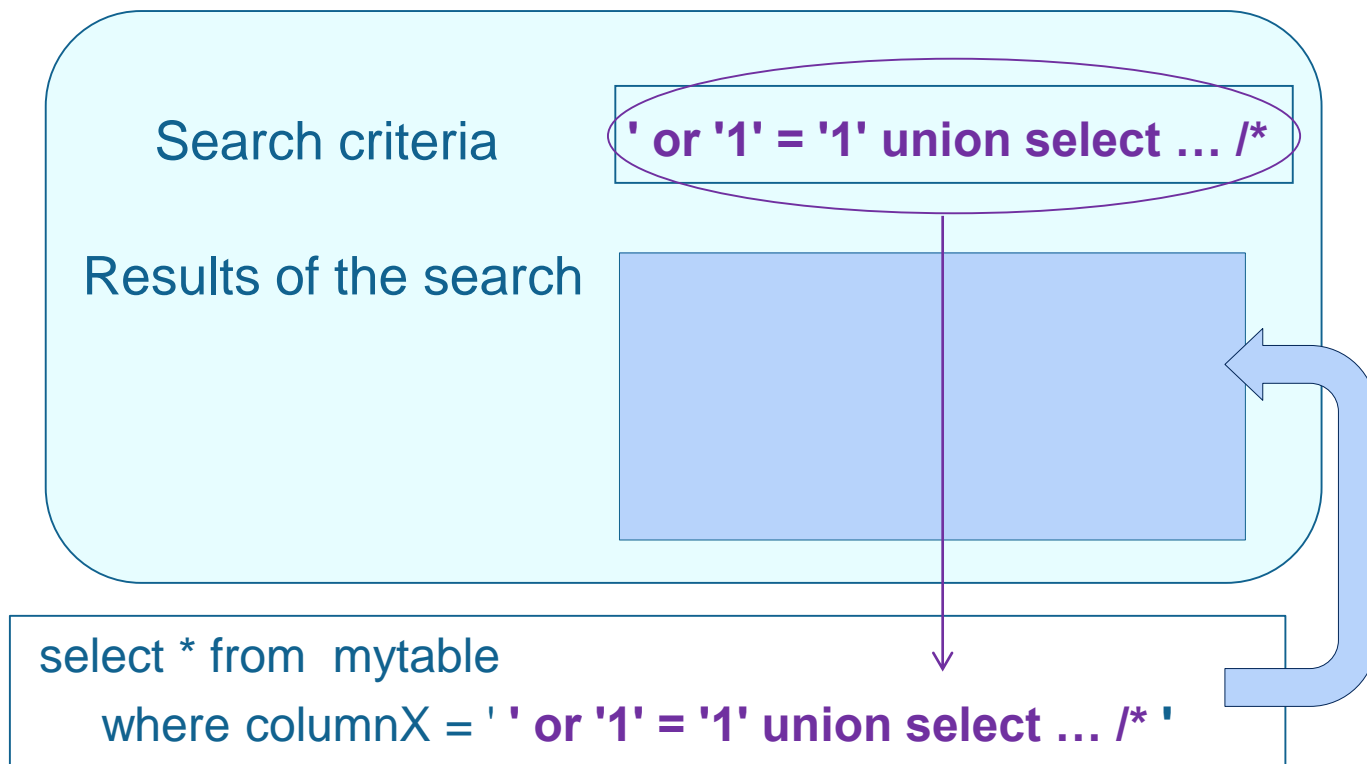
A light blue rounded rectangle represents a login form. It contains two labels: 'Login' and 'Password'. To the right of 'Login' is a text input field containing the SQL payload: `' ; insert into ... /*`. This payload is enclosed in a blue oval. A blue arrow points from this oval down to the SQL query in the block below.

```
select * from usertable
  where login = ' ' ; insert into ... /* ' and password = ' ... ' ;
```

Commentaire

Injection SQL

- Pour lire des données dans une base de données
 - Exemple



Injection SQL

Comment éviter les injections SQL?

Empêcher la mauvaise interprétation des '

Exemples :

- Utiliser des fonctions qui encapsulent les ' (ex: addslashes())
- Utiliser la classe **PreparedStatement** <> Statement

Accès aux bases de données

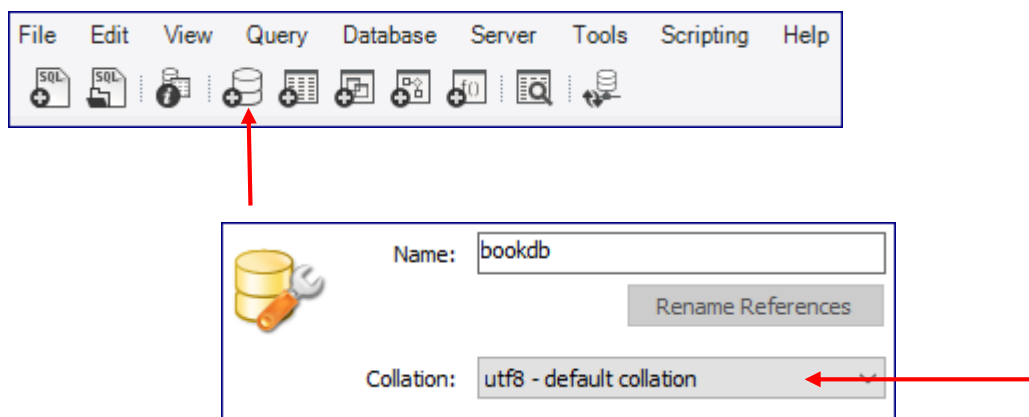
1. Injection SQL
2. Base de données MySQL

Serveur MySQL

- Installer un serveur MySQL
 - + Workbench
- Vérifier que le serveur tourne
 - Task Manager ⇒ onglet Services
 - ⇒ *MySQL56* : clic droit ⇒ Start

Schéma de base de données

- Créer un schéma de base de données en MySQL



- Sélectionner le schéma par défaut
 - Clic droit sur le schéma ⇒ Set as Default Schema

Table Book

book
<u>isbn</u>
pages [0..1]
title [0..1]
editiondate [0..1]

VARCHAR(20)

INT(6)

VARCHAR(100)

DATE

Table Book

Créer la table book

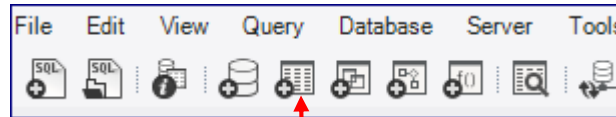


Table Name: Schema: **bookdb**

Collation: Engine:


Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
isbn	VARCHAR(20)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
pages	INT(6)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
title	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
editiondate	DATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Couche Model - Classe Book

```
public class Book
{ private String isbn;
  private Integer pagesNb;
  private String title;
  private GregorianCalendar editionDate;
  ...
}
```

```
private int pagesNB;
```



Prévoir des variables d'instance de type référence
et non de type primitif,
afin de gérer les attributs/colonnes facultatifs

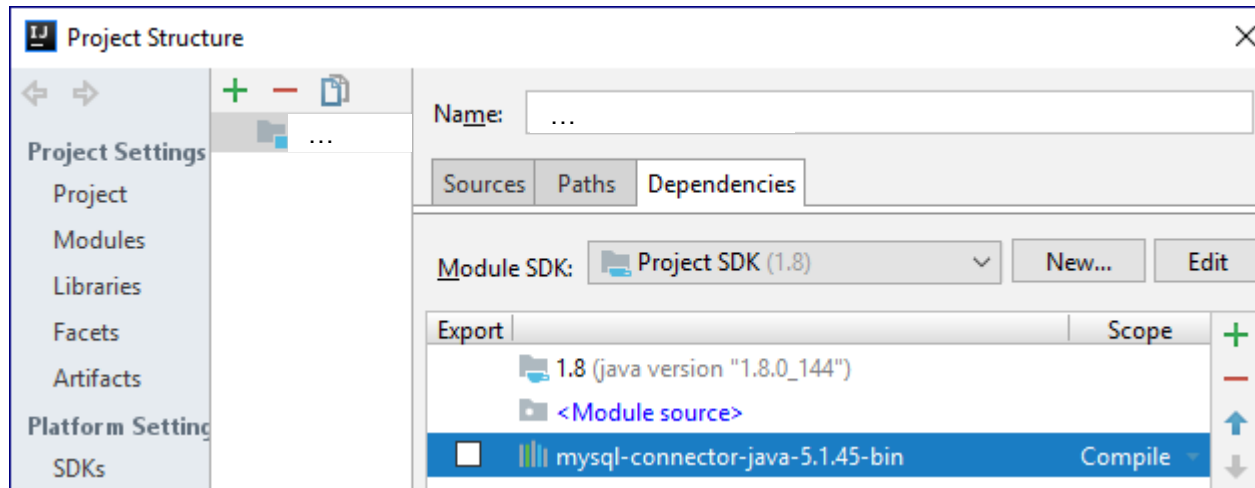
Accès aux bases de données

1. Injection SQL
2. Base de données MySQL
3. Accès aux bases de données à partir de java

- Java Database Connectivity
 - Application programming interface (API)
 - Pour Java
 - Définit comment accéder à une base de données

MySQL Connector/J

- MySQL fournit des drivers standard for JDBC
 - ⇒ Downloader connector/J mysql :
<https://dev.mysql.com/downloads/connector/j/>
- Ajouter le jar (.bin) dans les librairies du projet
 - File ⇒ Project Structure ⇒ Modules ⇒ Dependencies ⇒ +



Accès aux bases de données

1. Injection SQL
2. Base de données MySQL
3. Accès aux bases de données à partir de java
4. Création d'une connexion à une base de données

Ouvrir une connexion

couche **Data Access**

```
import java.util.* ;  
import java.sql.*;
```

Ouvrir une connexion

- Créer un objet de la classe **Connection**
 - Via la classe **DriverManager**
 - Méthode **getConnection**
 - Arguments
 - **Nom de la base de données (schéma)**
 - **Nom de l'utilisateur**
 - **Mot de passe**
 - Si l'objet est créé, la connexion est établie

Ouvrir une connexion

- Exemple

Connection connection =

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/bookdb",  
                             "root",  
                             "...") ← Password
```

- N.B : Si pas de connexion SSL

Connection connection =

```
DriverManager.getConnection(  
    "jdbc:mysql://localhost:3306/bookdb?useSSL=false", "...", "...")
```

Singleton Pattern

Objectif du pattern *singleton*

Garantir qu'une classe n'a qu'une seule instance

Comment créer un **objet unique** (une seule instance d'une classe) ?



Variable de classe privée (private **static**)

+

Constructeur privé (private)

+

Méthode (**static**) **getInstance()** qui retourne l'unique instance

Singleton Pattern

```
public class Singleton {  
    private static Singleton uniqueInstance;  
  
    // Autres variables d'instance  
  
    private Singleton (...) {...}  
  
    public static Singleton getInstance( ) {  
        if (uniqueInstance == null)  
            { ...  
                uniqueInstance = new Singleton(...) ; }  
        return uniqueInstance;  
    }  
    ...  
    // Autres méthodes  
}
```

Singleton Pattern

Adaptation

```
public class ClassX {  
    ...  
    private static ClassY uniqueInstance;  
    public static ClassY getInstance( ) {  
        if (uniqueInstance == null)  
            { ... }           // Créer une instance de ClassY  
        return uniqueInstance;  
    }  
    ...                       // Autres méthodes  
}
```

2 classes différentes

Même classe

Singleton Connection

Utilisation

Singleton singleton = **Singleton.getInstance()** ;

Cas d'utilisation dans le travail de fin d'année

stockage de l'objet Connection :

on ne crée la connexion que quand c'est nécessaire;

+ on ne crée qu'une fois la connexion

User Interface

MainJFrame

NewBookPanel

AllBooksPanel

AllBooksModel

Model

Controller

ApplicationController

Book

Business Logic

BookManager

Data Access

BookDBAccess

SingletonConnection

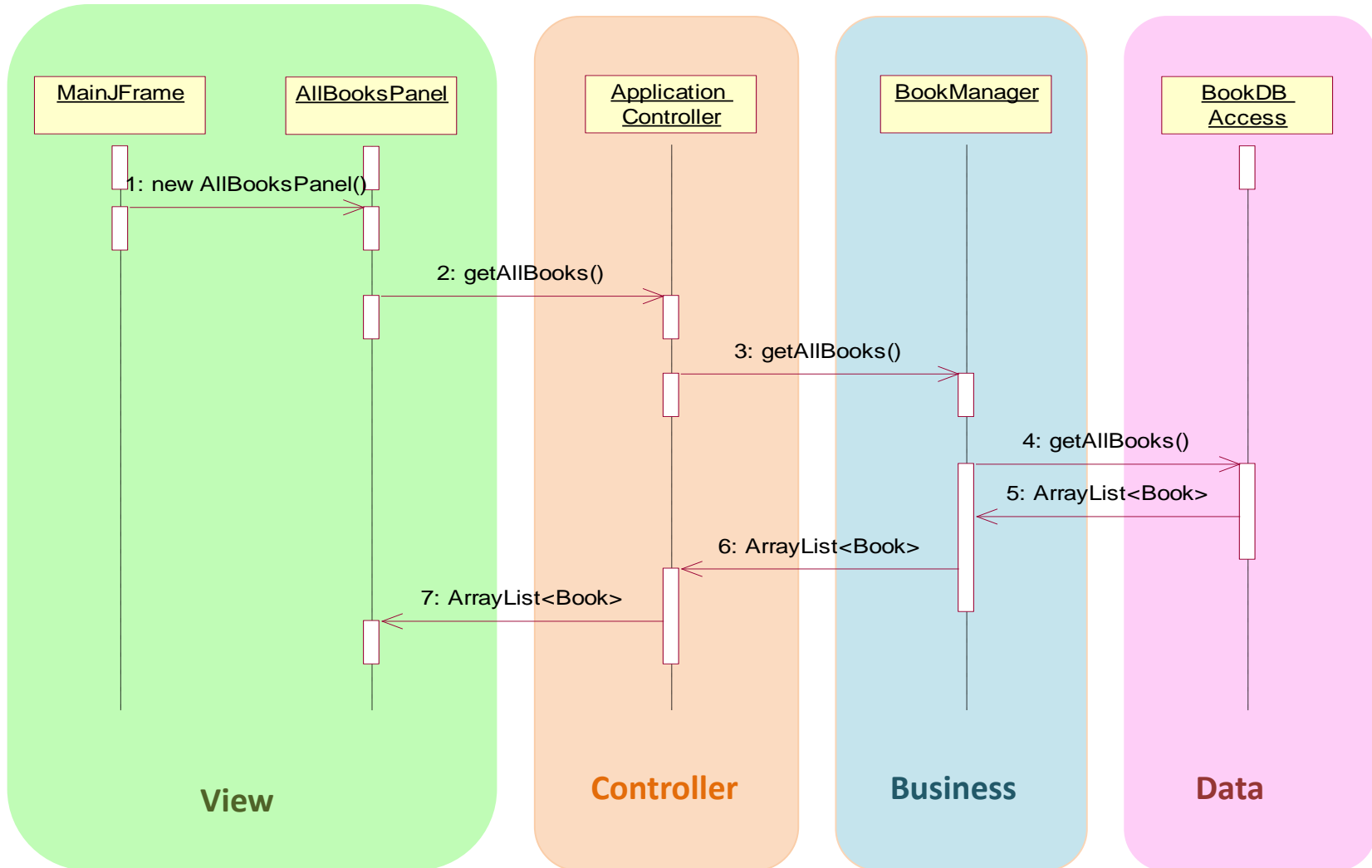
Singleton Connection

Gestion de la connexion unique

```
public class SingletonConnection {  
  
    private static Connection uniqueConnection;  
  
    public static Connection getInstance( ) ...  
  
        { if (uniqueConnection == null)  
  
            { ... // Essayer de créer une connexion à la base de données }  
  
        return uniqueConnection;  
    }  
  
}
```

Classe existante \Rightarrow impossible à modifier
 \Rightarrow Obligation de créer une autre classe
pour stocker et gérer le singleton
Connection

Exemple : Afficher la liste de tous les livres



Singleton Connection

```
public class BookDBAccess {  
  
    public ArrayList <Book> getAllBooks( ) throws AllBooksException  
  
    {  
        // Essayer d'accéder à la base de données  
        ↪ via SingletonConnection.getInstance( )  
  
        // Essayer de lire les livres dans la table Book  
  
        // Créer et retourner une ArrayList de livres  
  
    }  
}
```

Fermer une connexion

```
Connection connection = ... ;
```

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

↳ *throws **SQLException***

Accès aux bases de données

1. Injection SQL
2. Base de données MySQL
3. Accès aux bases de données à partir de java
4. Création d'une connexion à une base de données
5. Instructions DML : insert - delete - update

Insertion de lignes

Connection **connection** = ...; *Contient l'instruction SQL d'insertion à exécuter*

String **sql** = "insert ...values (? , ? , ? ...)";

PreparedStatement **statement** = **connection**.prepareStatement(**sql**);

statement.setString(1, "blabla"); —→ *Si colonne1 est de type chaîne de caractères*

statement.setInt(2, 100); —→ *Si colonne2 est de type entier*

java.sql.Date sqlDate = ...;

statement.setDate(3, sqlDate); —→ *Si colonne3 est de type date*

...

Insertion de lignes

Retourne le nombre de lignes modifiées



```
int insertedLinesNumber = statement.executeUpdate( );
```

↳ *throws **SQLException***

SQLInjection

N.B.

L'utilisation de la classe **PreparedStatement** empêche les injections SQL

<>

La classe **Statement**

Insertion de valeurs inconnues

Attention aux **valeurs inconnues** éventuelles (valeur **null** en SQL) lors d'insertion dans les **colonnes facultatives** !

Version 1

① **insert** pour les colonnes obligatoires

Book book = ... ;

*Insertion dans la table d'une ligne avec des valeurs dans toutes les colonnes **obligatoires***

String sql = "**insert** into book (isbn) values (?);"

PreparedStatement statement = connection.prepareStatement(sql);

statement.setString(1, book.getIsbn());

statement.executeUpdate();

Insertion de valeurs inconnues

② **update** pour les colonnes facultatives

Valeur à insérer dans une colonne **facultative** de type numérique

```
if ( book.getPagesNb( ) != null )  
{ sql = "update book set pages = ? where isbn = ' " + book.getIsbn() + " ' ";  
  statement = connection.prepareStatement(sql);  
  statement.setInt(1, book.getPagesNb( ));  
  statement.executeUpdate( );  
}
```

Insertion de valeurs inconnues

② **update** pour les colonnes facultatives

Valeur à insérer dans une colonne **facultative** de type **date**

```
if ( book.getEditionDate() != null )  
{ sql = "update book set editiondate = ? where isbn = ' " + book.getIsbn() + " ' ";  
  statement = connection.prepareStatement(sql);  
  statement.setDate(1,new java.sql.Date(book.getEditionDate().getTimeInMillis()));  
  statement.executeUpdate();  
}
```

De type **GregorianCalendar**

Insertion de valeurs inconnues

Version 2

① **insert** de valeurs *null* dans les colonnes facultatives

Exemple: livre avec un nombre de pages, un titre et une date d'édition inconnus

```
Book book = ... ;
```

```
String sql = "insert into book (isbn, pages, title, editiondate) values (?, ?, ?, ?)";
```

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

```
statement.setString(1, book.getIsbn( ));
```

```
statement.setNull(2, Types.INTEGER);
```

```
statement.setNull(3, Types.VARCHAR);
```

```
statement.setNull(4, Types.TIMESTAMP);
```

```
statement.executeUpdate( );
```

Commit

Confirmer l'instruction DML

⇒ Rendre permanentes les modifications
(cf notion de transaction SQL : commit)

Par défaut : `connection.setAutoCommit(true)` ; ↗
Modifications permanentes dans la BD (automatiquement)

Si `connection.setAutoCommit(false)` ;

⇒ `statement.executeUpdate(instructionSQL);`

...

`connection.commit();`

⇒ *Modifications permanentes dans la BD*

↗ *throws **SQLException***

Accès aux bases de données

1. Injection SQL
2. Base de données MySQL
3. Accès aux bases de données à partir de java
4. Création d'une connexion à une base de données
5. Instructions DML : insert - delete – update
6. Accès en lecture : select

Exécuter un select

Connection **connection** = ... ;

→ Contient la requête SQL à exécuter

String **sql** = " select ... where colonne1 = ? and colonne2 = ? and colonne3 = ? ...";

PreparedStatement **statement** = **connection**.prepareStatement(**sql**);

statement.setString(1, "blabla");

→ Si colonne1 est de type chaîne de caractères

statement.setInt(2, 100);

→ Si colonne2 est de type entier

java.sql.Date **sqlDate** = ...;

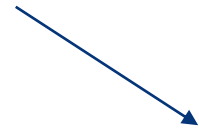
statement.setDate(3, sqlDate);

→ Si colonne3 est de type date

Récupérer une collection d'objets

```
ResultSet data = statement.executeQuery( );
```

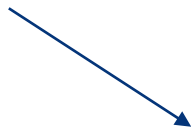
↳ throws *SQLException*



Un ResultSet contient les lignes de résultat de la requête

```
ResultSetMetaData meta = data.getMetaData( );
```

↳ throws *SQLException*



Un ResultSetMetaData contient des meta données sur le résultat de la requête

Récupérer une collection d'objets

Meta données = informations/renseignements sur les données

Exemples :

ResultSetMetaData **meta** = ... ;

meta.getColumnCount() —————> *Donne le nombre de colonnes*

meta.getColumnName(i) —————> *Donne le nom de la colonne i*

meta.getColumnType(i) —————> *Donne le type de la colonne i*

Récupérer une collection d'objets

Boucler sur les lignes de résultats de la requête (ResultSet) :

```
ResultSet data = ... ;
```

```
while ( data.next() )
```

```
{
```

```
    ...  Récupérer la ligne courante du ResultSet
```

```
}
```

Récupérer une collection d'objets

Boucler sur les lignes de résultats de la requête (ResultSet) :

```
ResultSet data = ... ;
```

```
while ( data.next( ) )
```

```
{ ...
```

```
data . { getString(...)  
        getInt(...)  
        getDate(...)  
        ...  
    }
```

Indice de la colonne ou
nom de la colonne

Types de colonnes <> classes Java

Type de colonne	Constante	Valeur	get	⇒ type de retour
Texte	Types. VARCHAR	12	getString()	⇒ <u>S</u>tring
Octet	Types. TINYINT	-6	getInt()	⇒ int
Entier	Types. SMALLINT	5	getInt()	⇒ int
Entier long	Types. INTEGER	4	getInt()	⇒ int
Réel simple	Types. REAL	7	getDouble()	⇒ <u>d</u>ouble
Réel double	Types. DOUBLE	8	getDouble()	⇒ <u>d</u>ouble
Décimal	Types. NUMERIC	2	getDouble()	⇒ <u>d</u>ouble
Oui/Non	Types. BIT	-7	getBoolean()	⇒ <u>b</u>oolean
Date/Heure	Types. TIMESTAMP	93	getDate ()	⇒ java.sql. <u>D</u>ate
Mémo	Types. LONGVARCHAR	-1	getString()	⇒ <u>S</u>tring

Types Date

Conversion `GregorianCalendar` ⇔ `java.sql.Date`

`GregorianCalendar` `calendar` = ... ;

`java.sql.Date` `sqlDate` = new `java.sql.Date`(`calendar.getTimeInMillis()`);

Conversion `java.sql.Date` ⇔ `GregorianCalendar`

`java.sql.Date` `sqlDate` = ... ;

`GregorianCalendar` `calendar` = new `GregorianCalendar`();

`calendar.setTime`(`sqlDate`);

Récupérer une collection d'objets

ResultSet **data** = ... ; → Contient le résultat de la requête "select * from Book"

Book book; int pages; String title; java.sql.Date editionDate;

ArrayList<Book> allBooks = new ArrayList < >();

while (**data.next()**)

{ **book** = new Book (**data.getString**("isbn");

→ Colonne obligatoire

pages = **data.getInt**("pages");

if (! **data.isNull()**)

{ **book.setPagesNb**(pages); }

→ Colonne facultative

title = **data.getString**("title");

if (! **data.isNull()**)

{ **book.setTitle**(title); }

Récupérer une collection d'objets

Récupération d'un objet `GregorianCalendar` à partir d'une colonne de type `date` en `BD`

```
editionDate = data.getDate("editiondate") ;
```

De type `java.sql.Date`

```
if ( ! data.isNull( ) )
```

```
    { GregorianCalendar calendar = new GregorianCalendar();
```

```
        calendar.setTime(editionDate);
```

```
        book.setEditionDate(calendar);
```

```
    }
```

```
allBook.add(book);
```

```
}
```

Accès aux bases de données

1. Injection SQL
2. Base de données MySQL
3. Accès aux bases de données à partir de java
4. Création d'une connexion à une base de données
5. Instructions DML : insert - delete – update
6. Accès en lecture : select
7. TableModel

couche **User Interface**

TableModel

Créer un modèle de données (TableModel) à partir d'une collection d'objets

*Nécessaire pour pouvoir afficher le résultat d'une requête
dans un composant Swing (JTable)*

```
import javax.swing.table.*;  
import java.util.*;
```

Classe abstraite



Créer une sous-classe de **AbstractTableModel**
qui contient 2 variables d'instance :

- La liste des noms de colonnes
- La liste d'objets correspondant aux données

TableModel

Attention :

Absolument redéfinir les méthodes :

getColumnCount()

getRowCount()

getColumnName(int col)

getValueAt(int row, int col)

getColumnClass(int col)

Car appelées implicitement par Java pour afficher correctement les données dans une JTable

TableModel

```
public class AllBooksModel extends AbstractTableModel
```

```
{ private ArrayList<String> columnNames;  
  private ArrayList<Book> contents;
```

```
  public AllBooksModel(ArrayList<Book> books)  
  { columnNames = new ArrayList<>();  
    columnNames.add("Isbn");  
    columnNames.add("Titre");  
    columnNames.add("Date d'édition");  
    columnNames.add("Nombre de pages");  
    setContents(books);}
```

```
  public int getColumnCount( ) { return columnNames.size( ); }
```

```
  public int getRowCount( ) { return contents.size( ); }
```

```
  public String getColumnName(int column) { return columnNames.get(column); }
```

TableModel

```
public Object getValueAt (int row, int column)
{ Book book = contents.get(row);
  switch(column)
  {   case 0 : return book.getIsbn();
      case 1: return book.getTitle();
      case 2: { if (book.getEditionDate() != null)
                  return book.getEditionDate().getTime();
                else return null;
            }
      case 3 : return book.getNbPages();
      default : return null;
  }
}
```

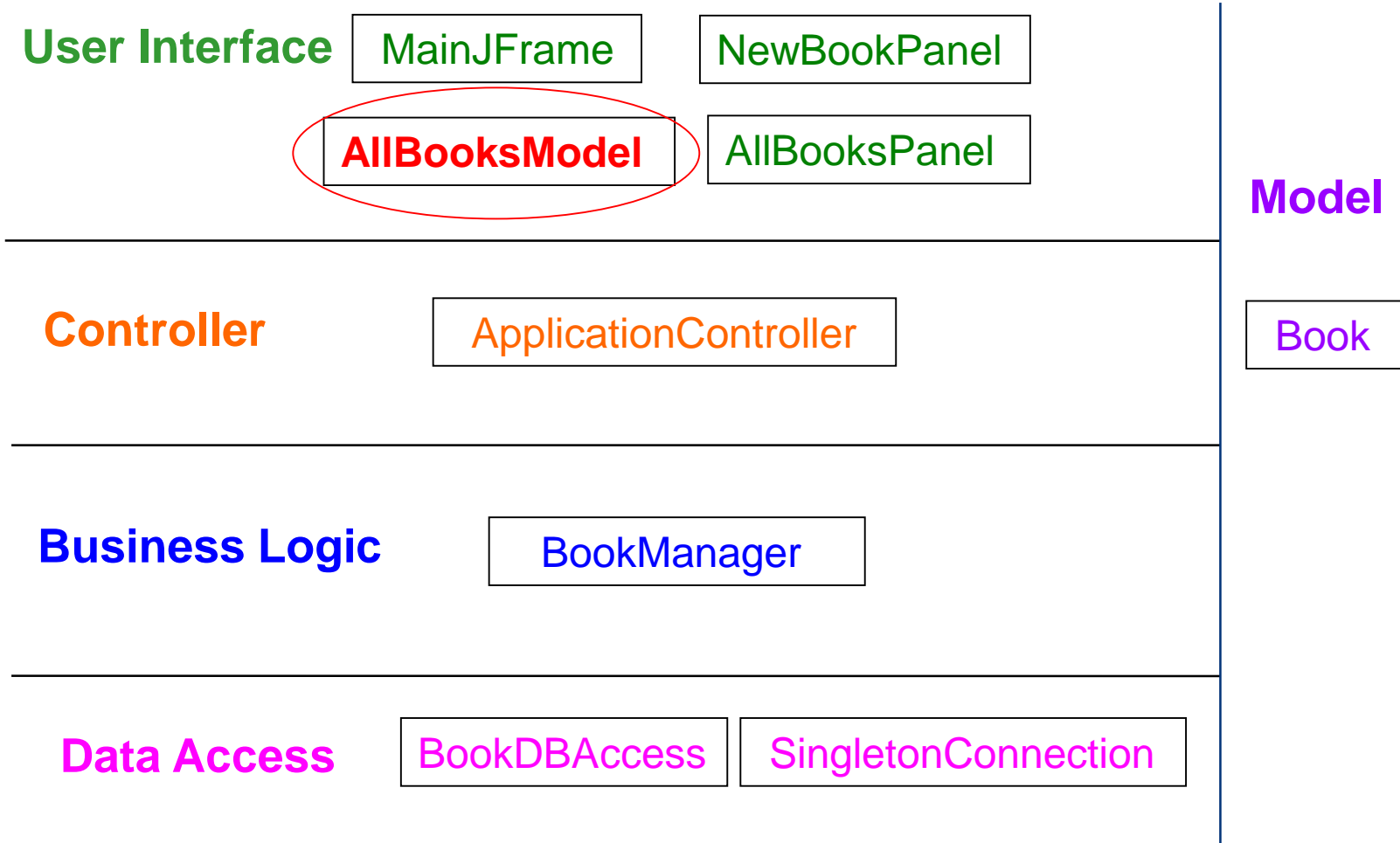
TableModel

Utilisée pour afficher correctement les colonnes en fonction de leur type

```
public Class getColumnClass (int column)
{
    Class c;
    switch (column)
    {
        case 0: c = String.class;
                break;
        case 1: c = String.class;
                break;
        case 2: c = Date.class;
                break;
        case 3: c = Integer.class;
                break;
        default: c = String.class;
    }
    return c;
}
```

...

Découpe en couches



Afficher un modèle de données

Créer une JTable

```
AllBooksModel model = ... ;
```

```
JTable table = new JTable (model) ;
```

Afficher une JTable

```
JScrollPane scrollPane = new JScrollPane (table) ;
```

✚ *A ajouter au container*

Choisir la taille des colonnes

`JTable table = ...`

```
TableColumn column = table.getColumnModel( ).getColumn(1);
```

```
column.setPreferredWidth(200); —————→ Imposer la largeur d'une colonne
```

```
table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
```



Empêche le redimensionnement automatique des colonnes

Récupérer une ligne sélectionnée

1. A la création de la JTable

```
JTable table = ...;
```

```
table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
ListSelectionModel listSelect = table.getSelectionModel( );
```

2. Dans la gestion d'événement *(ex: si clic sur bouton)*

```
int indiceLigneSelectionnee = listSelect.getMinSelectionIndex( );
```

...  *Récupération des valeurs de la ligne sélectionnée*