



Formateur

 **Pr. O. EL MIDAOUI**

HONORIS UNIVERSITIES – Professor & Software Engineer  
PhD in Web Geographic Information Retrieval GIR

## Modèle MVC : le modèle

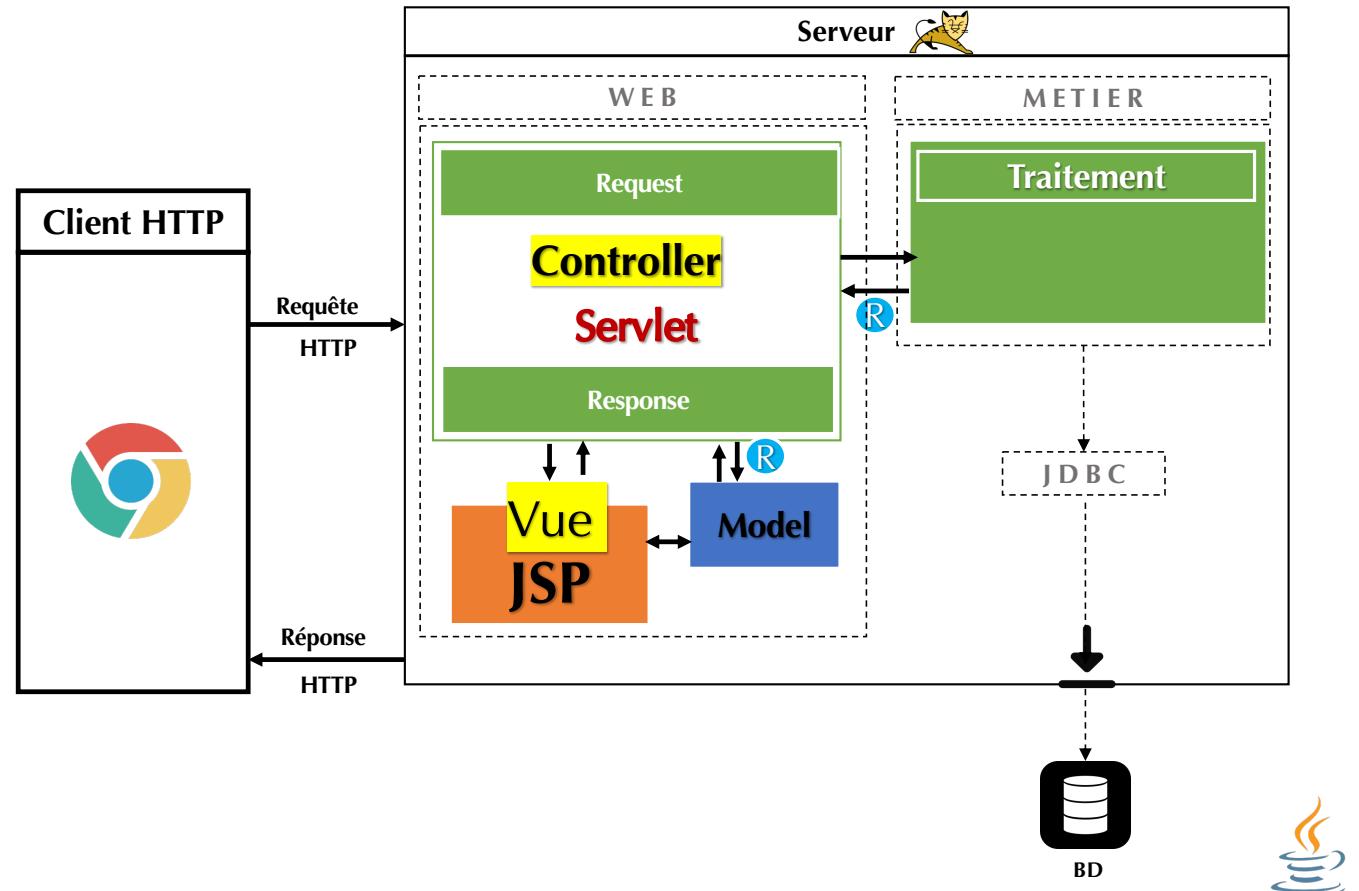
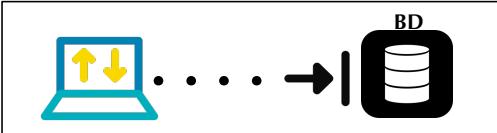
### Persister nos données d'application dans une BD [ORM]

#### L'existant

Maintenant on est capable de créer une application qui peut répondre à des requêtes HTTP toute en communiquant à une couche métier pour faire des traitements le stocker dans un modèle et puis générer du contenu dynamique qui sera retourner via une page JSP représentant la partie vue de notre application.

#### Objectif

L'étape qui suit c'est de pouvoir utiliser une **BD relationnel** pour pouvoir stocker nos données, et les récupérer au moment voulu



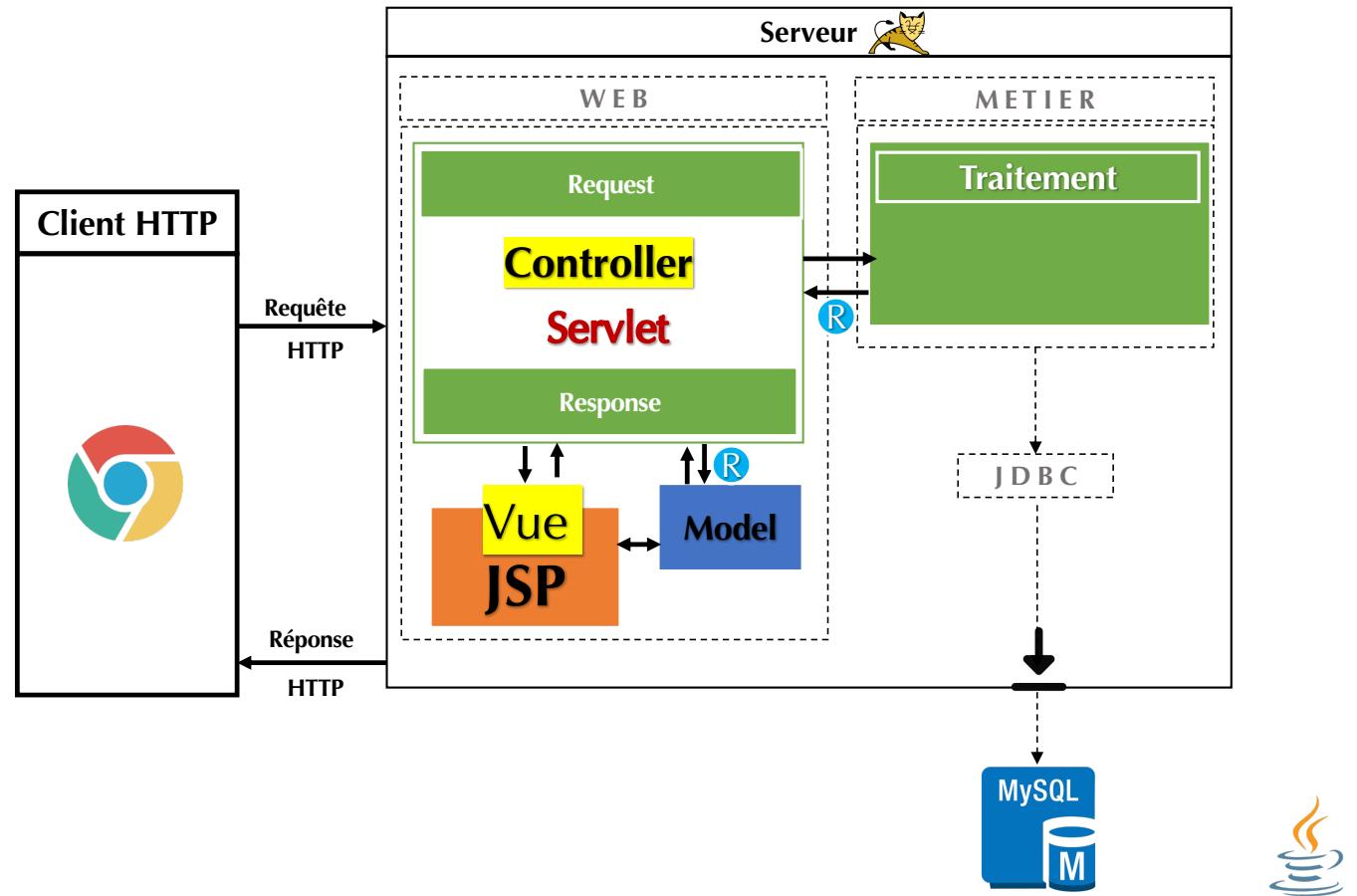
## Modèle MVC : le modèle

### Choix de La BD



La **BD relationnel** qu'on va utiliser est **MySQL**, puisque la plupart des application du marché l'utilise.

Noter bien que vous pouvez utiliser d'autre **SGBD** que le **MySQL**, c'est pas une obligation.



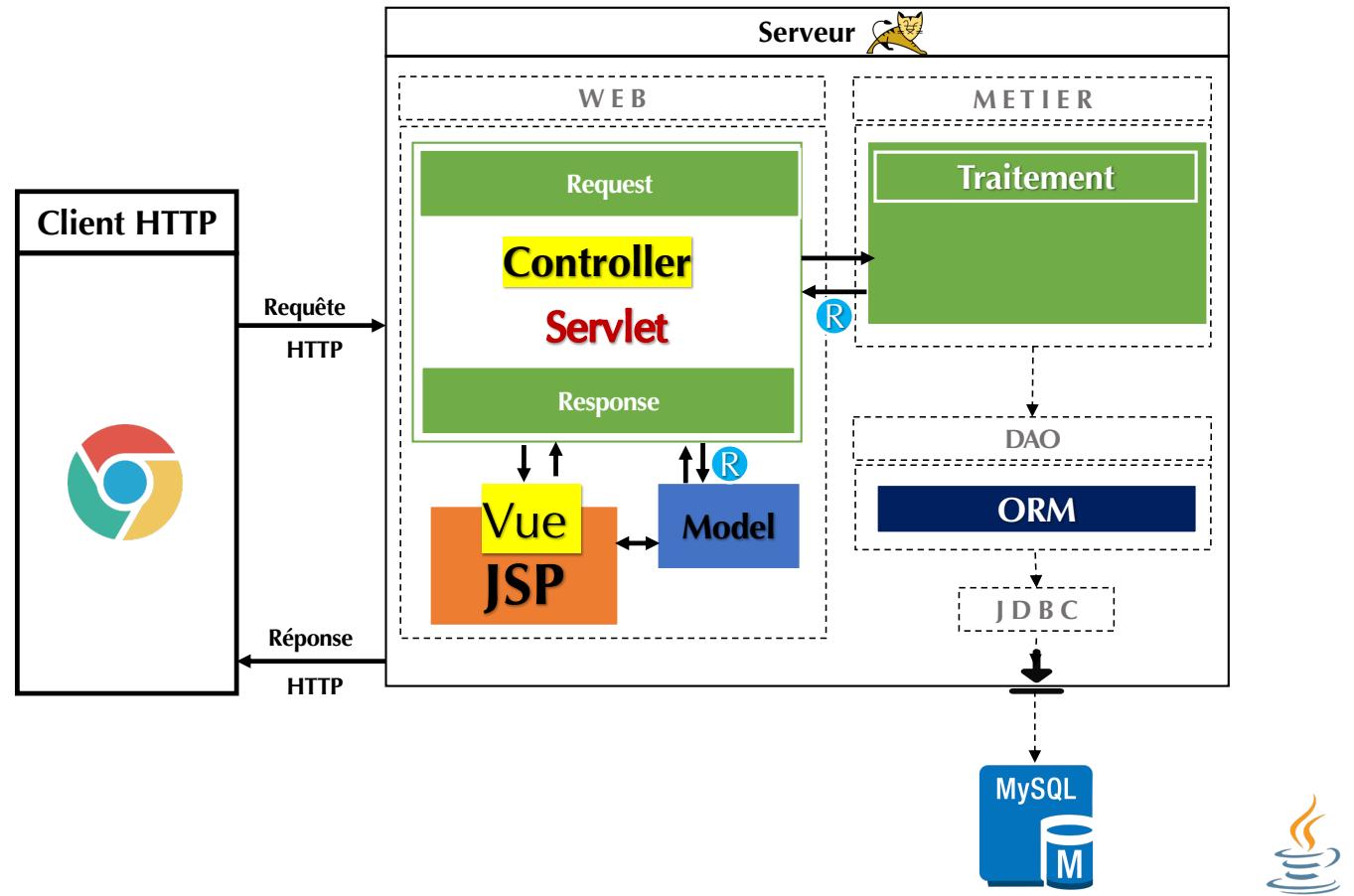
## Modèle MVC : le modèle

### Choix de La BD



La **BD relationnel** qu'on va utiliser est **MySQL**, puisque la plupart des application du marché l'utilise.

Noter bien que vous pouvez utiliser d'autre **SGBD** que le **MySQL**, c'est pas une obligation.



## Modèle MVC : le modèle

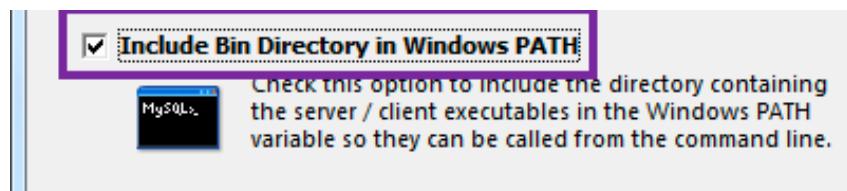
### Installation du SGBD MySQL



1. Rendez-vous sur [le site de téléchargement](#) de MySQL.
2. Sélectionnez le système d'exploitation sur lequel vous travaillez (Windows, Mac OS ou Linux)
3. Téléchargez la version de MySQL correspondante.

#### Sous Windows :

Laisser les choses par défaut.  
cochez juste l'option :  
"Include Bin Directory in Windows PATH"



#### Sinon après l'installation

Vérifiez le chemin vers MySQL dans votre explorateur : C:\Program Files\MySQL\MySQL Server 5.6\bin  
(le "bin" est important, c'est là que se trouvent les programmes clients).

Exécutez dans le terminal la commande :      `set PATH=%PATH%; C:\Program Files\MySQL\MySQL Server 5.6\bin`

#### Sous Linux :

Taper et exécuter la commande suivante  
dans le terminal :

```
sudo apt-get install mysql-server mysql-client.
```

Une fois l'installation terminée,  
Pour modifier le mot de passe par défaut de  
l'utilisateur "root" -> Taper :

```
sudo mysqladmin -u root -h localhost password 'entrez ici votre mot de passe'
```

#### Téléchargement du SGBD

#### Installation du SGBD



## Modèle MVC : le modèle

### Installation du SGBD MySQL



1. Téléchargez l'archive DMG qui vous convient (32 ou 64 bits)
2. Double-cliquez sur ce **.dmg** pour ouvrir l'image disque( contenant 4 fichiers dont deux **.pkg** )
3. Exécutez **mysql-5.5.9-osx10.6-x86\_64.pkg** (les chiffres peuvent changer selon la version de MySQL téléchargée, et suivez les instructions.
4. Après l'installation, vous pouvez ouvrir votre terminal et tapez les commandes suivantes :

```
cd /usr/local/mysql  
sudo ./bin/mysqld_safe
```

Entrez votre mot de passe si nécessaire puis tapez **Ctrl** + **Z**

Puis tapez la commande **bg** puis cliquer sur **Ctrl** + **D** : Quittez le terminal

#### Configuration

5. Par défaut, aucun mot de passe n'est demandé pour se connecter, même avec l'utilisateur **root** (qui a tous les droits). Pour définir un mot de passe pour cet utilisateur :

```
/usr/local/mysql/bin/mysqladmin -u root password votre_mot_de_passe
```

6. Pour accéder directement au logiciel client depuis la console sans devoir aller dans le dossier MySQL, il faut ajouter ce dossier à votre variable d'environnement **PATH**.

```
echo 'export PATH=/usr/local/mysql/bin:$PATH' >> ~/.profile
```

7. Redémarrez votre terminal pour que le changement prenne effet.

#### Installation du SGBD



## JDBC : Étape par Étape

Tout programme JDBC comprend les cinq étapes suivantes :

**Étape 1:** Création d'un objet **Connection**, pour se connecter au serveur de base de données

**Étape 2:** Création d'un objet **Statement**, sous la connexion créée précédemment, pour préparer nos requête SQL.

**Étape 3:** Formuler notre requête SQL et l'exécuter, via notre objet **Statement** déjà créé.

**Étape 4:** Traitement du résultat de la requête : s'il y a un résultat,  
on mappe le résultat stocké dans l'objet **ResultSet**. (**ORM**)

**Étape 5:** Fermeture des objets de **connexion** [**ResultSet**, **Statement** et **Connection**] pour libérer les ressources.

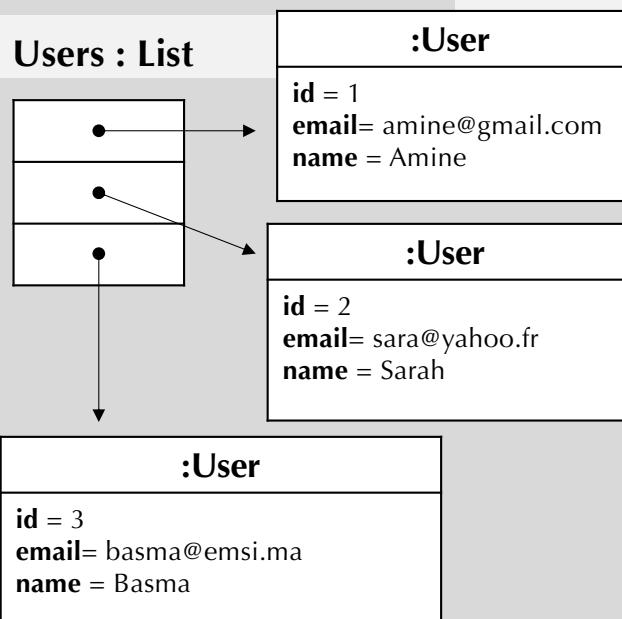
## Modèle MVC : ORM :: Mappage objet relationnel

### Oriented Object App

```
public class User {

    private Long id;
    private String name;
    private String email;

}
```



### Object Relational Mapping

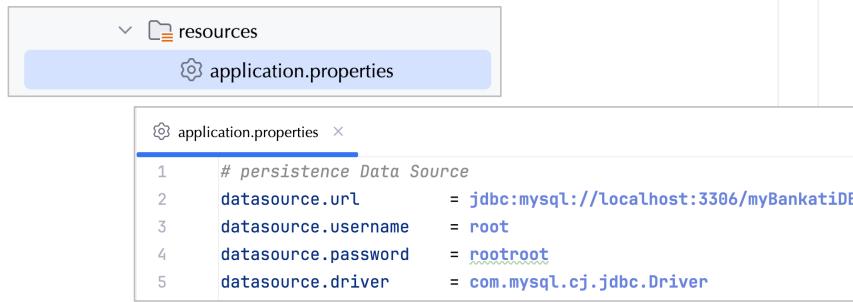
```
public List<User> findAll() {
    List<User> users = new ArrayList<>();
    Class.forName(className: "com.mysql.cj.jdbc.Driver");
    Connection cnx = DriverManager.getConnection
        (url: "jdbc:mysql://localhost:3306/myBankatiDB",
         user: "root", password: "rootroot");
    PreparedStatement ps = cnx.prepareStatement(sql: "SELECT * FROM utilisateur");
    ResultSet rs = ps.executeQuery();
    while (rs.next()) {
        var user = new User();
        user.setId(rs.getLong(columnLabel: "id"));
        user.setName(rs.getString(columnLabel: "name"));
        user.setEmail(rs.getString(columnLabel: "email"));
        users.add(user);
    }
    return users;
}
```

**Table utilisateur**  
**SGBD MySQL, BD : MyBankatiDB**

| ID | Email           | Name  |
|----|-----------------|-------|
| 1  | amine@gmail.com | Amine |
| 2  | sara@yahoo.fr   | Sarah |
| 3  | basma@emsi.ma   | Basma |

# JDBC : Étape par Étape

## Création du Singleton de la connexion



The screenshot shows a file tree with a 'resources' folder containing an 'application.properties' file. The file contains the following configuration:

```
1 # persistence Data Source
2 datasource.url      = jdbc:mysql://localhost:3306/myBankatiDB
3 datasource.username  = root
4 datasource.password = rootroot
5 datasource.driver   = com.mysql.cj.jdbc.Driver
```

```
public class SessionFactory { 5 usages
{
    private static SessionFactory instance = new SessionFactory(); 1 usage
    private Connection session; 2 usages

    public static SessionFactory getInstance() { return instance; } no usages
    public Connection openSession() { return session; } no usages

    private SessionFactory(){ 1 usage
        try {
            var configFile = SessionFactory.class
                .getClassLoader()
                .getResourceAsStream( name: "application.properties");

            if(configFile != null){
                Properties properties = new Properties();
                properties.load(configFile);
                System.out.println("Properties file loaded successfully");

                var driverClass = properties.getProperty("datasource.driver");
                Class.forName(driverClass);
                System.out.println("DB Driver loaded successfully");

                // Chaine de connexion
                var url      = properties.getProperty("datasource.url");
                var username = properties.getProperty("datasource.username");
                var password = properties.getProperty("datasource.password");

                session = DriverManager.getConnection(url, username, password);
                System.out.println("Connection established successfully");

            } else System.err.println("Properties File not found ");
        }
    catch (SQLException e) { System.err.println("Connection Failed"); }
    catch (IOException e) { System.err.println("Could not load Properties File");}
    catch (ClassNotFoundException e) { System.err.println("Could not load DB driver"); }
}
}
```

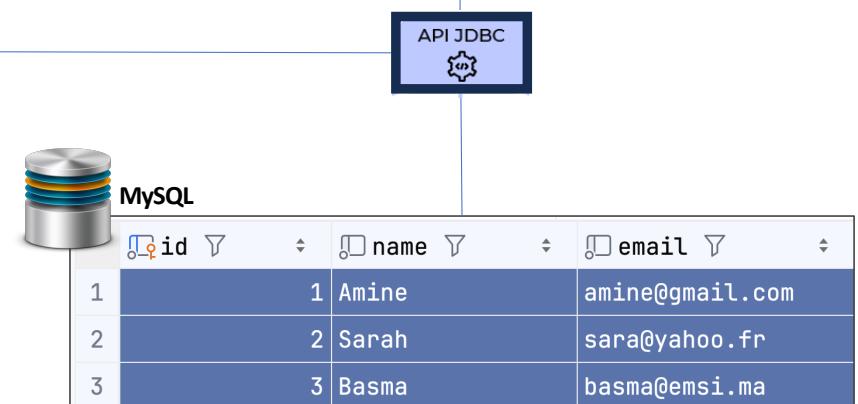
## JDBC : Étape par Étape

Mapping du résultat :

```
private User mapping (ResultSet rs) throws SQLException {  
  
    User user = null;  
  
    var id = rs.getLong (columnLabel: "id");  
    var name = rs.getString (columnLabel: "name");  
    var email = rs.getString (columnLabel: "email");  
  
    user = new User(id, name, email);  
  
    return user;  
}
```

```
public class User {  
  
    private Long id;  
    private String name;  
    private String email;  
}
```

Entité JavaBean



## JDBC : Étape par Étape

On peut créer une méthode utile pour la Préparation de la requête :

```
public static PreparedStatement initPS (Connection CNX,
                                         String SQL,
                                         boolean generateKey,
                                         Object... Columns)      throws SQLException {

    PreparedStatement PS = null;
    PS = CNX.prepareStatement(SQL,
                             generateKey ?
                             Statement.RETURN_GENERATED_KEYS :
                             Statement.NO_GENERATED_KEYS);

    for (int i = 0; i < Columns.length; i++)  PS.setObject(i+1, Columns[i]);
    return PS;
}
```

## JDBC : Étape par Étape

Pour la Fermeture des objets de connexion :

```
public static void closeDA00bjets(ResultSet RS) {
    try {
        if (RS != null)
            RS.close();
    } catch (SQLException e) {
        System.out.println("Problem while closing ResultSet :(" + e.getMessage());
    }
}

public static void closeDA00bjets(PreparedStatement PS) {

    try {
        if (PS != null)
            PS.close();
    } catch (SQLException e) {
        System.out.println("Problem while closing Statement :(" + e.getMessage());
    }
}

public static void closeDA00bjets(Connection cnx) {

    try {
        if (cnx != null)
            cnx.close();
    } catch (SQLException e) {
        System.out.println("Problem while closing Connection :(" + e.getMessage());
    }
}
```