18/05/2018

Nair Alic et Guillaume Zaretti

HEIG-VD

Laboratoire 4

SER

Table des matières

[1. Introduction 2](#_Toc516564928)

[2. Présenter le principe de votre solution et son architecture 2](#_Toc516564929)

[PlexWFC 2](#_Toc516564930)

[Main.java 2](#_Toc516564931)

[PlexAdmin 4](#_Toc516564932)

[controlleurWFC 4](#_Toc516564933)

[Applicatif PlexMedia 6](#_Toc516564934)

[3. Présenter un extrait représentatif des fichiers que vous aurez générés ou modifiés 8](#_Toc516564935)

[4. présenter quelques printscreens 8](#_Toc516564936)

[5. Conclusion 8](#_Toc516564937)

# Introduction

Dans le cadre de ce laboratoire, nous avons dû mettre en œuvre le mécanisme RMI pour la chaine Plex. Il est question de traiter et d’implémenter les échanges d’informations entre l’applicatif du World Film Center PlexWFC et l’applicatif PlexAdmin. Une couche de communication RMI va permettre l’échange d’information.

# Présenter le principe de votre solution et son architecture

## PlexWFC

### Main.java

**import** java.io.Serializable;  
**import** java.rmi.RemoteException;  
**import** java.sql.Wrapper;  
**import** java.util.Observable;  
**import** java.util.Observer;  
**import** java.util.Scanner;  
  
**import** ch.heigvd.iict.ser.rmi.IClientApi;  
**import** ch.heigvd.iict.ser.rmi.IServerApi;  
**import** db.MySQLAccess;  
**import** ch.heigvd.iict.ser.imdb.models.Data;  
  
**public class** Main **extends** Observable **implements** IServerApi {  
  
 **private** IServerApi **serverApi** = **null**;  
 **static** {  
 *// this will load the MySQL driver, each DB has its own driver* **try** {  
 Class.*forName*(**"com.mysql.jdbc.Driver"**);  
 } **catch** (ClassNotFoundException e) {  
 System.***err***.println(**"MySQL drivers not found !"**);  
 System.*exit*(1);  
 }  
  
 *//database configuration* MySQLAccess.*MYSQL\_URL* = **"docr.iict.ch"**;  
 MySQLAccess.*MYSQL\_DBNAME* = **"imdb"**;  
 MySQLAccess.*MYSQL\_USER* = **"imdb"**;  
 MySQLAccess.*MYSQL\_PASSWORD* = **"imdb"**;  
 }  
  
 **private static** Scanner *scanner* = **new** Scanner(System.***in***);  
  
 **public static void** main(String[] args) {  
 Main main = **new** Main();  
 main.run();  
 }  
  
 **private** Data **lastData** = **null**;  
  
 **private void** run() {  
  
 **boolean** continuer = **true**;   
 **while**(continuer) {  
 System.***out***.print(**"Select the data version to download [1/2/3/0=quit]: "**);  
 **int** choice = -1;  
 **try** {  
 choice = *scanner*.nextInt();  
 } **catch**(Exception e) {  
 e.printStackTrace();  
 }  
   
 **if**(choice == 0) continuer = **false**;  
 **else if**(choice >= 1 && choice <= 3) {  
 Worker worker = **new** Worker(choice);  
 **this**.**lastData** = worker.run();  
  
 *//****TODO notify client* if**(**lastData** != **null**){  
 setChanged();  
 notifyObservers(**lastData**);  
 System.***out***.println(**"new observers updated"**);  
 }  
   
 }  
 }  
 }  
  
 */\*\*  
 \* Method used by clients to register on the server  
 \*  
 \** ***@param client*** *The client  
 \** ***@throws*** *RemoteException  
 \*/* @Override  
 **public void** addObserver(IClientApi client) **throws** RemoteException {  
 WrappedObserver wrappedObserver = **new** WrappedObserver(client);  
 addObserver(wrappedObserver);  
 System.***out***.println(**"wrappedObserver added"**);  
 }  
  
 */\*\*  
 \* Method used by clients to check the connection with the server  
 \*  
 \** ***@return*** *true is the server is reachable  
 \** ***@throws*** *RemoteException  
 \*/* @Override  
 **public boolean** isStillConnected() **throws** RemoteException {  
 **return serverApi** != **null**;  
 }  
  
 */\*\*  
 \* Method used by clients to get all the data  
 \*  
 \** ***@return*** *The data  
 \** ***@throws*** *RemoteException  
 \*/* @Override  
 **public** Data getData() **throws** RemoteException {  
 **return lastData**;  
 }  
  
 **private class** WrappedObserver **implements** Observer, Serializable{  
 **private static final long *serialVersionUID*** = -2067345842536415833L;  
  
 IClientApi **clientApi** = **null**;  
  
 **public** WrappedObserver(IClientApi clientApi) {  
 **this**.**clientApi** = clientApi;  
 }  
  
 */\*\*  
 \* This method is called whenever the observed object is changed. An  
 \* application calls an <tt>Observable</tt> object's  
 \* <code>notifyObservers</code> method to have all the object's  
 \* observers notified of the change.  
 \*  
 \** ***@param o*** *the observable object.  
 \** ***@param arg*** *an argument passed to the <code>notifyObservers</code>  
 \*/* @Override  
 **public void** update(Observable o, Object arg) {  
 **try**{  
 **clientApi**.update(o.toString(), IClientApi.Signal.***UPDATE\_REQUESTED***, arg.toString());  
 }**catch**(RemoteException e){  
 System.***out***.println(**"Remote exception removing observer : "** + **this**);  
 o.deleteObserver(**this**);  
 }  
 }  
 }  
}

## PlexAdmin

### controlleurWFC

**package** controllers;  
  
**import** ch.heigvd.iict.ser.imdb.models.Data;  
**import** ch.heigvd.iict.ser.rmi.IClientApi;  
**import** ch.heigvd.iict.ser.rmi.IServerApi;  
**import** views.\*;  
  
**import** java.rmi.Naming;  
**import** java.rmi.RemoteException;  
**import** java.rmi.server.UnicastRemoteObject;  
  
**public class** ControleurWFC **extends** UnicastRemoteObject **implements** IClientApi {  
  
 **private** ControleurGeneral **ctrGeneral**;  
 **private static** MainGUI *mainGUI*;  
  
 **private** IServerApi **serverApi** = **null**;  
 Data **database** = **null**;  
 **private static final long *serialVersionUID*** = -8478788162368553187L;  
  
 */\*\*  
 \* Creates and exports a new UnicastRemoteObject object using an  
 \* anonymous port.  
 \* <p>  
 \* <p>The object is exported with a server socket  
 \* created using the {****@link*** *RMISocketFactory} class.  
 \*  
 \** ***@throws*** *RemoteException if failed to export object  
 \** ***@since*** *JDK1.1  
 \*/* **public** ControleurWFC(ControleurGeneral ctrGeneral, MainGUI mainGUI) **throws** RemoteException{  
 **this**.**ctrGeneral**=ctrGeneral;  
 ControleurWFC.*mainGUI*=mainGUI;  
 **try** {  
 IServerApi remoteServiceApi = (IServerApi) Naming.*lookup*(**"//localhost:9999/RmiService"**);  
 **this**.**serverApi**.addObserver(**this**);  
  
 } **catch** (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
  
  
 @Override  
 **public void** update(Object observable, Signal signalType, String updateMsg) **throws** RemoteException {  
 **database** = **serverApi**.getData();  
 **ctrGeneral**.initBaseDeDonneesAvecNouvelleVersion(**database**);  
 }  
}

## Applicatif PlexMedia

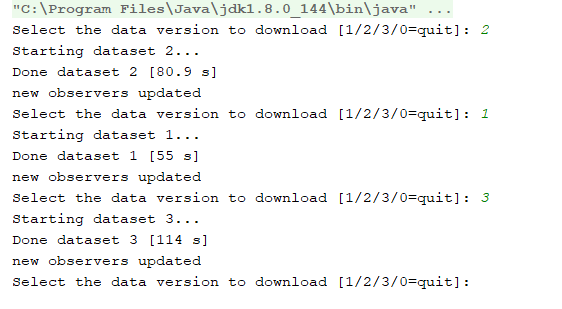
**import** ch.heigvd.iict.ser.imdb.models.Data;  
**import** ch.heigvd.iict.ser.rmi.IClientApi;  
**import** ch.heigvd.iict.ser.rmi.IServerApi;  
**import** javafx.application.Application;  
**import** javafx.event.ActionEvent;  
**import** javafx.event.EventHandler;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.Button;  
**import** javafx.scene.layout.StackPane;  
**import** javafx.stage.Stage;  
  
**import** java.net.MalformedURLException;  
**import** java.rmi.Naming;  
**import** java.rmi.NotBoundException;  
**import** java.rmi.RemoteException;  
**import** java.rmi.server.RMISocketFactory;  
**import** java.rmi.server.UnicastRemoteObject;  
  
**public class** Main **extends** Application {  
  
 **public static void** main(String[] args){  
 System.***out***.println(**"start PlexMedia"**);  
 *launch*(args);  
 }  
  
 */\*\*  
 \* The main entry point for all JavaFX applications.  
 \* The start method is called after the init method has returned,  
 \* and after the system is ready for the application to begin running.  
 \* <p>  
 \* <p>  
 \* NOTE: This method is called on the JavaFX Application Thread.  
 \* </p>  
 \*  
 \** ***@param primaryStage*** *the primary stage for this application, onto which  
 \* the application scene can be set. The primary stage will be embedded in  
 \* the browser if the application was launched as an applet.  
 \* Applications may create other stages, if needed, but they will not be  
 \* primary stages and will not be embedded in the browser.  
 \*/* **public void** start(Stage primaryStage) **throws** Exception {  
  
 primaryStage.setTitle(**"GUIPlexMedia"**);  
 Button button= **new** Button(**"getProjections"**);  
  
 button.setOnAction(**new** EventHandler<ActionEvent>() {  
 **public void** handle(ActionEvent e) {  
 System.***out***.println(**"click btn\_getProjections"**);  
 RmiClient rmiClient = **null**;  
 **try** {  
 rmiClient = **new** RmiClient((IServerApi) Naming.*lookup*(**"//localhost:4321/RmiService"**));  
 } **catch** (RemoteException e1) {  
 e1.printStackTrace();  
 } **catch** (NotBoundException e1) {  
 e1.printStackTrace();  
 } **catch** (MalformedURLException e1) {  
 e1.printStackTrace();  
 }  
 **try** {  
 rmiClient.getProjectionsFromPlexAdmin();  
 } **catch** (RemoteException e1) {  
 e1.printStackTrace();  
 }  
  
 }  
 });  
  
 StackPane layout= **new** StackPane();  
 layout.getChildren().add(button);  
  
 Scene scene1= **new** Scene(layout, 300, 250);  
 primaryStage.setScene(scene1);  
  
 primaryStage.show();  
  
 }  
  
  
  
 **private class** RmiClient **extends** UnicastRemoteObject **implements** IClientApi {  
 **private** IServerApi **serverApi** = **null**;  
 **private** Data **data** = **null**;  
  
 */\*\*  
 \* Creates and exports a new UnicastRemoteObject object using an  
 \* anonymous port.  
 \* <p>  
 \* <p>The object is exported with a server socket  
 \* created using the {****@link*** *RMISocketFactory} class.  
 \*  
 \** ***@throws*** *RemoteException if failed to export object  
 \** ***@since*** *JDK1.1  
 \*/* **protected** RmiClient(IServerApi serverApi) **throws** RemoteException {  
 **try** {  
 **this**.**serverApi** = serverApi;  
 **this**.**serverApi**.addObserver(**this**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 **public** Data getProjections() **throws** RemoteException {  
 **try** {  
 **data** = **serverApi**.getData();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return data**;  
 }  
  
 **public void** getProjectionsFromPlexAdmin() **throws** RemoteException {  
 **try** {  
 System.***out***.println(getProjections().getJsonData());  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 **public void** update(Object observable, Signal signalType, String updateMsg) **throws** RemoteException {  
  
 }  
 }  
}

# Présenter un extrait représentatif des fichiers que vous aurez générés ou modifiés

Nous n’avons par réussi à générer ou modifier les fichier, nous n’avons pas réussi à faire fonctionner l’architecture.

# présenter quelques printscreens

WFC



# Conclusion

Pour conclure ce laboratoire, nous n’avons malheureusement pas réussi a exécuter configurer correctement l’application pour la faire fonctionner en réseau avec la base de donnée. Nous estimons avoir tout de même bien compléter le code et implémenter les fonctions demandées. On c’est bien basé sur le cours du prof et les exemples fourni pour implémenter RMI mais qu’on a pas réussi à faire fonctionner Plex.