

|  |
| --- |
|  |
| **Vinci Quiz**  Réalisation |
|  |

Page de service

**Référence :** Vinci Quiz

**Plan de classement :** analyse-conception-quiz

**Niveau de confidentialité :** confidential

**Mises à jour**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Auteur | Description du changement |
| 1.0.0 | 12-04-2016 | Rodriguez Thomas  Cucherat Julian  Ponson Nathan | Création Networking Inc.  Standalone PC |
| 2.0.0 | 11-04-2023 | Rodriguez Thomas  Cucherat Julian  Ponson Nathan  Celio Poncet | Développement de la BDD et du serveur.  Multiplayer |

**Validation**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Nom | Rôle |
| 1.0.0 | 18-12-2022 | Delphine TALARON | Direction Technique  Vinci Quiz Project |

**Diffusion**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Nom | Rôle |
| 1.0.0 | 18-12-2022 | All | SLAM Networking Inc. |
|  |  |  |  |

Objet du document

Ce document décrit l'implémentation Java du projet Vinci Thermo Green.

Ce projet vise à produire une application réalise l'implémentation du diagramme des classes métier et du diagramme de séquence objet présenté dans la documentation d'analyse conception.

Sommaire

[Page de service 0](#_Toc133046378)

[Objet du document 0](#_Toc133046379)

[Sommaire 1](#_Toc133046380)

[1 Architecture 2](#_Toc133046381)

[1.1 Prérequis 3](#_Toc133046382)

[2 Implémentation des classes métiers 4](#_Toc133046383)

[3 Accès aux données 5](#_Toc133046384)

[3.1 Lire un fichier PROPERTIES 6](#_Toc133046385)

[3.2 Lire et écrire dans une base de données – JDBC 6](#_Toc133046386)

[3.2.1 Se connecter à une base de données 7](#_Toc133046387)

[3.2.2 Envoyer une requête SQL 8](#_Toc133046388)

[3.2.3 Manipuler le résultat 8](#_Toc133046389)

[4 Gérer une collection d'objet 8](#_Toc133046390)

[5 Interface Homme Machine 10](#_Toc133046391)

[5.1 Gérer plusieurs JPanel dans une JFrame 10](#_Toc133046392)

[5.1.1 Layout 11](#_Toc133046393)

[5.2 Divers composants "atomiques" 11](#_Toc133046394)

[5.2.1 Les boutons pour valider les saisies 13](#_Toc133046395)

[5.2.2 Les boutons radio pour la sélection de la réponse 16](#_Toc133046396)

[5.2.3 Les listes déroulantes 17](#_Toc133046397)

[6 Cryptographie 18](#_Toc133046398)

[6.1 Différence entre Hachage et Cryptage 18](#_Toc133046399)

[6.2 Hachage : jBCrypt 19](#_Toc133046400)

[6.2.1 Implementation 20](#_Toc133046401)

[6.3 Chiffrement – déchiffrement : Jasypt 21](#_Toc133046402)

[6.3.1 Implémentation 21](#_Toc133046403)

[7 Jeu multijoueur : serveur websocket 22](#_Toc133046404)

[7.1 KryoNet 23](#_Toc133046405)

[7.2 Implémentation 24](#_Toc133046406)

[7.2.1 Connexion / Déconnexion 24](#_Toc133046407)

[7.2.2 Envoi / Réception d’un message 25](#_Toc133046408)

[8 JAR - archivage - déploiement et signature 25](#_Toc133046409)

[8.1 Le package JAR archive 25](#_Toc133046410)

[8.1.1 Génération d’un jar file assistée par Eclipse 26](#_Toc133046411)

[8.2 Le package JAR runnable 27](#_Toc133046412)

[8.2.1 Génération d’un jar file assistée par Eclipse 27](#_Toc133046413)

[9 TABLE DES ILLUSTRATIONS 27](#_Toc133046414)

# Architecture

Conformément à l'analyse conception, la réalisation de l'application est structurée en trois couches selon une structure qui ressemble à un design-pattern MVC (Model View Controler) mais qui en réalité ne l'est pas vraiment. Cependant cette structure du code permet d'envisager dans une version ultérieure une évolution vers un modèle réellement MVC.

Le modèle MVC fonctionne selon le principe illustré par le schéma ci-dessous[[1]](#footnote-1) :

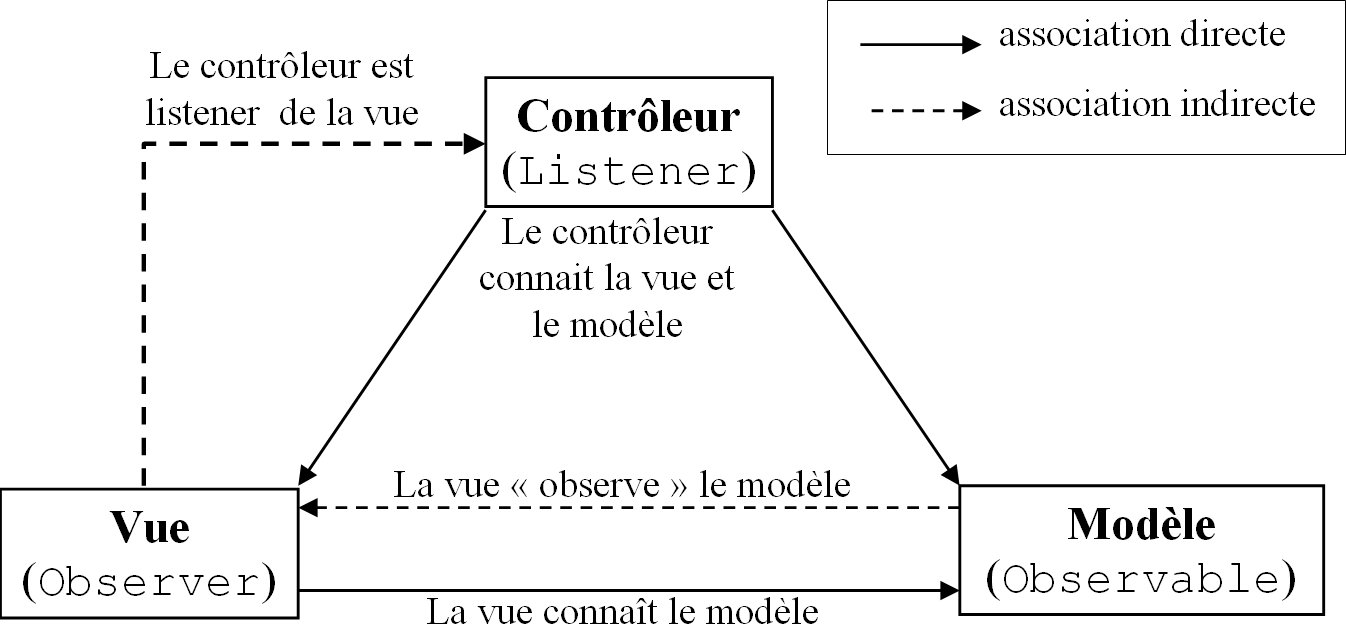


Figure : principe de fonctionnement du modèle MVC

La conception de la v.1.0.0 de l'application Vinci Quiz prévoit la mise en œuvre d'un contrôleur. Ce contrôleur représente non pas le "listener" au sens MVC du terme mais le Data Access Object (DAO)[[2]](#footnote-2) d'une architecture n-tiers.

Cependant, l'utilisation de la bibliothèque graphique Swing permet d'écouter la vue au sens propre du design-pattern MVC. Cela pourra être abordé lors d'une version ultérieure.

L'utilisation d'un DAO permet de s'abstraire de la façon dont les données sont stockées au niveau des objets métier. Ainsi, le changement du mode de stockage ne remet pas en cause le reste de l'application. Seules les classes dites "techniques" seront à modifier.

Les objets en mémoire vive sont liés à des données persistantes (stockées en base de données, dans des fichiers, dans des annuaires, etc…). Le modèle DAO regroupe les accès aux données persistantes dans des classes techniques spécifiques, plutôt que de les disperser. Il s'agit surtout de ne pas écrire ces accès dans les classes "métier", qui ne seront modifiées que si les règles de gestion métier changent.

Ce modèle complète le modèle MVC (Modèle - Vue - Contrôleur), qui préconise de séparer dans des classes les différentes problématiques :

* Des "vues" (charte graphique, ergonomie)
* Du "modèle" (cœur du métier)
* Des "contrôleurs" (tout le reste : l'enchaînement des vues, les autorisations d'accès, etc…)

Ci-dessous est présenté le diagramme des composants mettant en application le modèle MVC :

Une image contenant diagramme

Description générée automatiquement

Figure : diagramme des composants de l’application

Dans ce schéma, on observe clairement le modèle MVC. Dans le package controller, la classe Start est le lancement de l’application, elle contient le main. Ce dernier va instancier un objet controller qui va, comme son nom l’indique, contrôler l’application :

public static void main(String[] args) {

//controller instantiation

Controller theController = new Controller();

}

Ce contrôleur est celui qui « voit tout », c’est-à-dire que c’est par lui que vont passer les différents composants de l’application pour communiquer entre eux. Il faut pour cela créer des constructeurs pour chaque classe qui attendent un objet controller en paramètre. Lors de son initialisation, le contrôleur va donc instancier la plupart des objets en passant comme paramètre, lui :

**public** Controller(){

}

## Prérequis

Pour commencer la création d’un projet évolutif de cette ampleur, avoir une architecture solide est primordial. C’est pour cela que nous créons un Build. Un espace de travail composé de différents dossiers prévus à certaines fonctions définies comme accueillir la documentation, les tests de codes etc.

Le Build est composé des dossiers essentiels comme nous pouvons l’observer ci-dessous.

Une image contenant texte

Description générée automatiquement

Figure : dossiers essentiels du Build

# Implémentation des classes métiers

L'analyse a permis de modéliser les classes métiers selon le diagramme ci-dessous (non documenté, cf. document d'analyse-conception) :

Une image contenant diagramme

Description générée automatiquement

Figure : diagramme des classes métiers

L'application lit une base de données qui contient les questions et réponses du quiz. La classe "Game" stocke les questions et réponses de la partie, ainsi que les joueurs…

public class Game {

private int idGame;

private String name;

private int idLeader;

private int nbQuestion;

private ArrayList<Player> playerList;

private ArrayList<Question> groupQuestions;

private String time;

private ArrayList<Connection> lesConnections;

private int statusGame;

public Game() {};

public Game(String name, int idLeader, int nbQuestion, Player leader, String time) {

this.name = name;

this.idLeader = idLeader;

this.nbQuestion = nbQuestion;

this.playerList = new ArrayList<Player>();

this.playerList.add(leader);

this.time = time;

}

public Game(String name, int idGame, int nbQuestion, String time) {

this.name = name;

this.nbQuestion = nbQuestion;

this.time = time;

this.idGame = idGame;

}

public Game(int idGame, String name, int idLeader, ArrayList<Player> playerList, ArrayList<Question> theQuestions, int nbQuestion, String time) {

this.idGame = idGame;

this.name = name;

this.idLeader = idLeader;

this.playerList = playerList;

this.groupQuestions = theQuestions;

this.nbQuestion = nbQuestion;

this.time = time;

}

public Game(int idGame, String name, int idLeader, ArrayList<Player> playerList, ArrayList<Question> theQuestions, int nbQuestion, String time, Connection connection) {

this.idGame = idGame;

this.name = name;

this.idLeader = idLeader;

this.playerList = playerList;

this.groupQuestions = theQuestions;

this.nbQuestion = nbQuestion;

this.lesConnections = new ArrayList<Connection>();

this.lesConnections.add(connection);

this.time = time;

this.statusGame = 1;

}

public int getIdGame() {

return idGame;

}

public void setIdGame(int idGame) {

this.idGame = idGame;

}

[…]

}

# Accès aux données

## Lire un fichier PROPERTIES[[3]](#footnote-3)

Un fichier de propriétés se compose de paires clé-valeur de types de string qui peuvent avoir n'importe quelle extension, bien que .properties est recommandé pour les distinguer facilement des autres fichiers. Un fichier de propriétés permet aux technologies compatibles de stocker les paramètres de configuration d'un logiciel. Ils sont également utilisés afin de stocker les chaînes de caractères standardisées, ils sont connus comme des Property Resource Bundles.

Nous pouvons lire les fichiers de propriétés en Java en utilisant le Properties classer. La Properties classe représente un ensemble persistant de propriétés qui peuvent être chargées à partir d'un flux à l'aide de son load() méthode :

Properties properties = **new** Properties();

**try** (InputStream fis =

getClass().getClassLoader().getResourceAsStream("data/conf.properties")) {

properties.load(fis);

}

Afin de lire les données souhaitées, il faut utiliser la méthode getProperty("") :

properties.getProperty("jdbc.driver.class")));

## Lire et écrire dans une base de données – JDBC[[4]](#footnote-4)

Dans l'application Vinci Quiz, on utilise une base de données permettant de stocker les questions et leurs réponses. Afin de récupérer les informations de la base, nous devons nous connecter à cette dernière et exécuter différentes requêtes.

JDBC (Java DataBase Connectivity) est une API (Application Programming Interface) java disponible depuis la version 1.1 du JDK. Cette API est constituée d'un ensemble d'interfaces et de classes qui permettent l'accès, à partir de programmes java, à des données tabulaires. Pour pouvoir utiliser JDBC, il faut un pilote qui est spécifique à la base de données à laquelle nous voulons accéder.

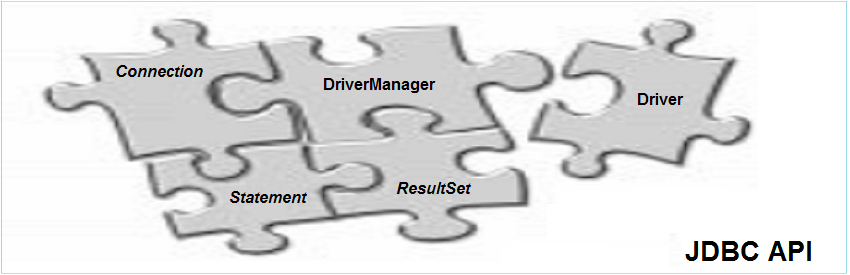


Figure : principe de fonctionnement de l'API JDBC

Les composants de l'Api JDBC se composent essentiellement de :

1. DriverManager : classe utilisée pour gérer la liste de Driver (database drivers).
2. Driver : interface utilisée pour relier la base de données avec des liens. Une fois le driver téléchargé, le programmeur ne doit pas l’appeler ouvertement.
3. Connection : interface avec toutes les méthodes de communication avec la base de données. L'objet de connexion représente le contexte de communication. Toutes les communications avec la base donnée se font par l'objet de Connexion uniquement.
4. Statement : incarne une déclaration SQL qui est envoyée à la base donnée pour analyser, compiler, planifier et mettre en œuvre.
5. ResultSet : l’ensemble des éléments récupérés par l’exécution de la requête.

### Se connecter à une base de données

Le schéma ci-dessous montre les façons d’accéder à une base de données :

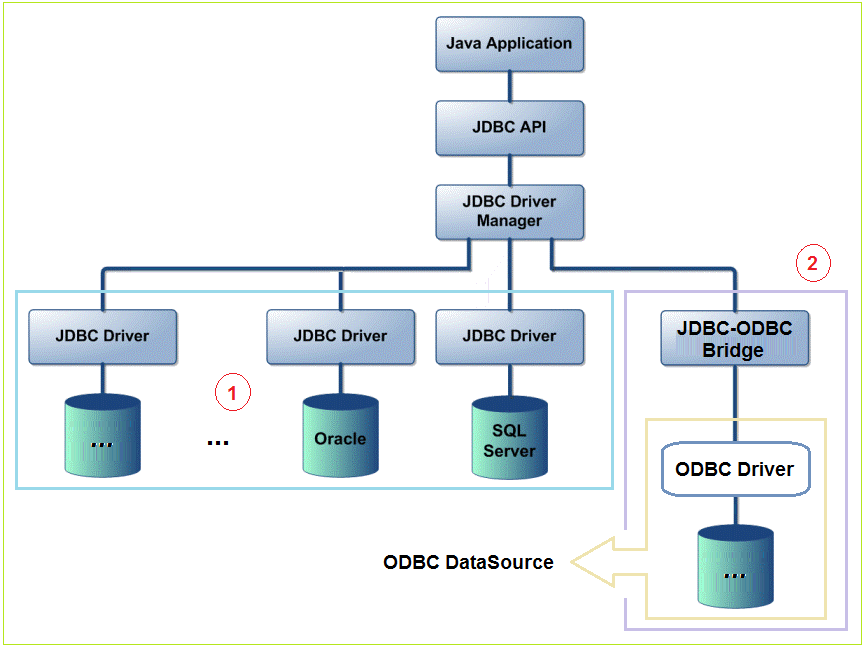


Figure : les différentes méthodes de connexion avec JDBC

Il existe deux façons de travailler avec une base de données spécifique :

* Fournir les types de bases donnés à la bibliothèque Driver, c'est la façon directe. Si une DB oracle (ou autre DB) est utilisée, il faut télécharger les bibliothèques pour le type DB. (option 1)
* Déclarer un "ODBC DataSource" et utiliser le pont JDBC-ODBC pour se connecter à la "ODBC DataSource" autre. Le pont JDBC-ODBC est disponible dans JDBC API. (option 2)

Dans notre cas, on utilise une base de données MySQL, on a donc à faire à l’option 1 :

Nous devons d’abord charger le driver, il faut utiliser la méthode forName de la classe Class :

Class.forName(com.mysql.cj.jdbc.Driver);

Nous accédons par la suite à la base via un URL qui spécifie :

* l'utilisation de JDBC
* le driver ou le type du SGBDR
* l'identification de la base

Exemple :

urlCnx = properties.getProperty("jdbc.url");

loginCnx = properties.getProperty("jdbc.login");

passwordCnx = properties.getProperty("jdbc.password");

Ouverture de la connexion :

conn = DriverManager.*getConnection*(urlCnx, loginCnx, passwordCnx);

Pour finir, nous devons créer un Statement. java.sql.Statement est un canal de communication établi sur une connexion. Elle dispose d'une méthode executeQuery qui prend en argument une requête sql de type select et renvoie un Objet de type ResultSet.

Statement st = conn.createStatement();

### Envoyer une requête SQL

Il existe trois types d’exécutions de requête :

* executeQuery() : pour les requêtes qui retournent un ResultSet
* executeUpdate() : pour les requêtes INSERT, UPDATE, DELETE, CREATE TABLE et DROP TABLE
* execute() : pour quelques cas rares (procédures stockées)

Il est attendu dans les parenthèses, la requête SQL à exécuter.

Avec la méthode executeQuery() :

String strSqlQuestion = "select \* from answer where id\_question = " + id + " order by rand()";

ResultSet resultSet = st.executeQuery(strSqlQuestion);

### Manipuler le résultat

La méthode executeQuery() renvoi un ResultSet. Ce dernier peut être assimilé à une table de données représentant un ensemble de résultats de base de données. Le ResultSet se parcourt itérativement row par row. Les colonnes sont référencées par leur numéro ou par leur nom. L'accès aux valeurs des colonnes se fait par les méthodes getXXX() où XXX représente le type de l'objet.

**for** (**int** i = 0; i < 4; i++) {

resultSet.next();

String indexA = Integer.*toString*(i + 1);

String descA = resultSet.getString(1);

Boolean resA = resultSet.getBoolean(2);

answers.add(**new** Answer(indexA, descA, resA));

}

# Gérer une collection d'objet[[5]](#footnote-5)

Les collections sont des objets qui permettent de gérer des ensembles d'objets. Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, etc…

Une collection est un regroupement d'objets qui sont désignés sous le nom d'éléments.

L'API Collections propose un ensemble d'interfaces et de classes dont le but est de stocker de multiples objets. Elle propose quatre grandes familles de collections, chacune définie par une interface de base :

* List : collection d'éléments ordonnés qui accepte les doublons
* Set : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons
* Map : collection sous la forme d'une association de paires clé/valeur
* Queue et Deque : collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement

Dans l'application Vinci Quiz, on utilise une collection pour les objets "Question" instanciés à partir des valeurs lues dans la base de données afin de stocker toutes les questions d’un quiz.

**[…]**

public ArrayList<Question> getQuestions(int nbQuestion) {

ArrayList<Question> questions = new ArrayList<Question>();

String query = "call get\_game(?)";

try (Connection connection = DriverManager.getConnection(urlCnx, loginCnx, passwordCnx);

CallableStatement statement = connection.prepareCall(query);) {

statement.setInt(1, nbQuestion);

statement.execute();

ResultSet resultSet = statement.getResultSet();

while (resultSet.next()) {

ArrayList<Answer> answers = new ArrayList<Answer>();

int idQuestion = resultSet.getInt("id\_question");

String nomQuestion = resultSet.getString("desc\_question");

for (int i = 1; i < 5; i++) {

String codeAnswer = Integer.toString(i);

String descAnswer = resultSet.getString("desc\_answer" + i);

Boolean resAnswer = resultSet.getBoolean("is\_correct" + i);

answers.add(new Answer(codeAnswer, descAnswer, resAnswer));

}

// mélange la liste des réponses

Collections.shuffle(answers);

questions.add(new Question(idQuestion, nomQuestion, answers));

}

} catch (Exception e) {

e.printStackTrace();

}

return questions;

}

**[…]**

Pour filtrer la collection, on ne modifie pas la collection en supprimant les objets hors critère et ceci pour au moins deux raisons :

Techniquement, modifier une collection pendant son parcours nécessite des précautions sinon on obtient ce genre d'erreur :

Exception in thread "AWT-EventQueue-0" java.util.ConcurrentModificationException

Pour cause, les index ne sont plus cohérents entre l'itérateur et la collection elle-même.

Notez bien : la comparaison entre deux chaînes de caractère à ne pas confondre avec la comparaison entre deux pointeurs sur deux chaines.[[6]](#footnote-6)

La classe String implémente l'interface java.lang.Comparable, et possède donc une méthode "compareTo(String s)" renvoyant :

* un nombre négatif si la chaîne actuelle est placée avant la chaîne passée en paramètre;
* zéro (0) si les deux chaînes sont strictement égales;
* un nombre positif si la chaîne actuelle est placée après la chaîne passée en paramètre.

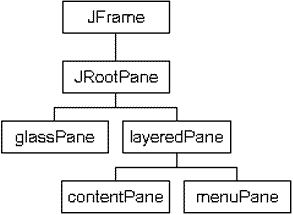
int comparaison = "Hello".compareTo("World");

System.out.println(comparaison); /\* Nombre négatif car H < W \*/

# Interface Homme Machine

## Gérer plusieurs JPanel dans une JFrame[[7]](#footnote-7)

Une application graphique doit avoir un composant top-level comme composant racine, c'est-à-dire un composant qui inclut tous les autres composants.

Dans Swing, il y a 3 composants top-level:

1. JFrame,
2. JDialog,
3. JApplet.

Les composants top-level possèdent un content pane qui contient tous les composants visibles d’un top-level. Un composant top-level peut aussi contenir une barre de menu

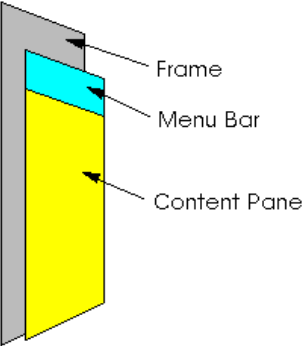
Une JFrame est une fenêtre avec un titre et une bordure.

Chaque JFrame de Swing possède une "vitre", un container racine dans lequel on peut poser tous les autres composants.

Tous les composants associés à un objet JFrame sont gérés par un objet de la classe JRootPane. Un objet JRootPane contient plusieurs Panes. Tous les composants ajoutés au JFrame doivent être ajoutés à un des Pane du JRootPane et non au JFrame directement. C'est aussi à un de ces Panes qu'il faut associer un layout manager si nécessaire. Le Layout manager par défaut du contentPane est BorderLayout.

Le JRootPane se compose de plusieurs éléments :

* glassPane : par défaut, le glassPane est un JPanel transparent qui se situe au-dessus du layeredPane. Le glassPane peut être n'importe quel composant : pour le modifier il faut utiliser la méthode setGlassPane() en fournissant le composant en paramètre.
* layeredPane qui se compose du contentPane (un JPanel par défaut) et du menuBar (un objet de type JMenuBar). Le contentPane est par défaut un JPanel opaque dont le gestionnaire de présentation est un BorderLayout. Ce panel peut être remplacé par n'importe quel composant grâce à la méthode setContentPane().



Container pane = monIHM.getContentPane();

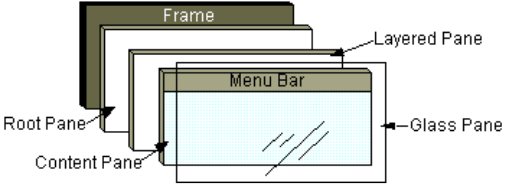
Les conteneurs intermédiaires sont utilisés pour structurer l’application graphique.

Le composant top-level contient des composants conteneur intermédiaires.

Un conteneur intermédiaire peut contenir d’autres conteneurs intermédiaires

Swing propose plusieurs conteneurs intermédiaires :

* JPanel : le conteneur intermédiaire le plus neutre.
* JScrollPane : offre des ascenseurs, il permet de visionner un composant plus grand que lui.
* JSplitPane : est un panel coupé en deux par une barre de séparation.
* JTabbedPane : permet d’avoir des onglets.
* JToolBar : est une barre d'icones.
* etc…

Swing offre également des conteneurs Intermédiaires spécialisés qui offrent des propriétés particulières aux composants qu’ils accueillent :

* JRootPane est obtenu à partir d’un top-level et composé de:
  + glass pane
  + layered pane
  + content pane
  + menu bar
* JLayeredPane permet de positionner les composants dans un espace à trois dimensions
* JInternalFrame permet d’afficher des petites fenêtres dans une fenêtre.

Swing fournit des composants atomiques qui sont les éléments d'interaction de l'IHM :

* boutons, CheckBox, Radio
* Combo box
* List, menu
* TextField, TextArea, Label
* FileChooser, ColorChooser,
* etc…

### Layout[[8]](#footnote-8)

Pour placer des composants dans un container, Java utilise le "Layout".

Un layout est une entité Java qui place les composants les uns par rapport aux autres.

Le layout réorganise les composants lorsque la taille du container varie.

Il y a plusieurs layouts :

* AbsoluteLayout qui permet de placer les composant par leu coordonnées (x,y).
* BorderLayout sépare un container en cinq zones: NORTH, SOUTH, EAST, WEST et CENTER.
* BoxLayout empiler les composants du container verticalement ou horizontalement.
* CardLayout permet d’avoir plusieurs conteneurs les uns au-dessus des autres (comme un jeu de cartes).
* FlowLayout range les composants sur une ligne. Si l’espace est trop petit, une autre ligne est créée. Le FlowLayout est le layout par défaut des JPanel.
* GridLayout positionne les composants sur une grille.
* GridBagLayout place les composants sur une grille, mais des composants peuvent être contenus dans plusieurs cases. Pour exprimer les propriétés des composants dans la grille, on utilise un GridBagConstraints. Un GridBasConstraints possède :
  + gridx, gridy pour spécifier la position.
  + gridwidth, gridheight pour spécifier la place.
  + fill pour savoir comment se fait le remplissage.

Un layout n’est pas contenu dans un container, il gère le positionnement des composants à l'intérieur du container.

## Divers composants "atomiques"

L'IHM de l'application est composée de plusieurs JPanel ayant chacun des informations à afficher en fonction de l’avancement du quiz. Prenons comme exemple le JPanel de connexion, où il est demandé le nom du joueur et son mot de passe. Une fois le bouton "Connexion" appuyé, si la connexion est réussie, une fonction NextPanel() va être appelé et le JPanel va être retiré de la JFrame et mis à null, et le JPanel suivant va être initialisé puis ajouter à la JFrame. Chaque JPanel devant être affiché à donc une classe spécifique étendue de JPanel.

Une image contenant diagramme

Description générée automatiquement

Figure : JPanel permettant de se connecter

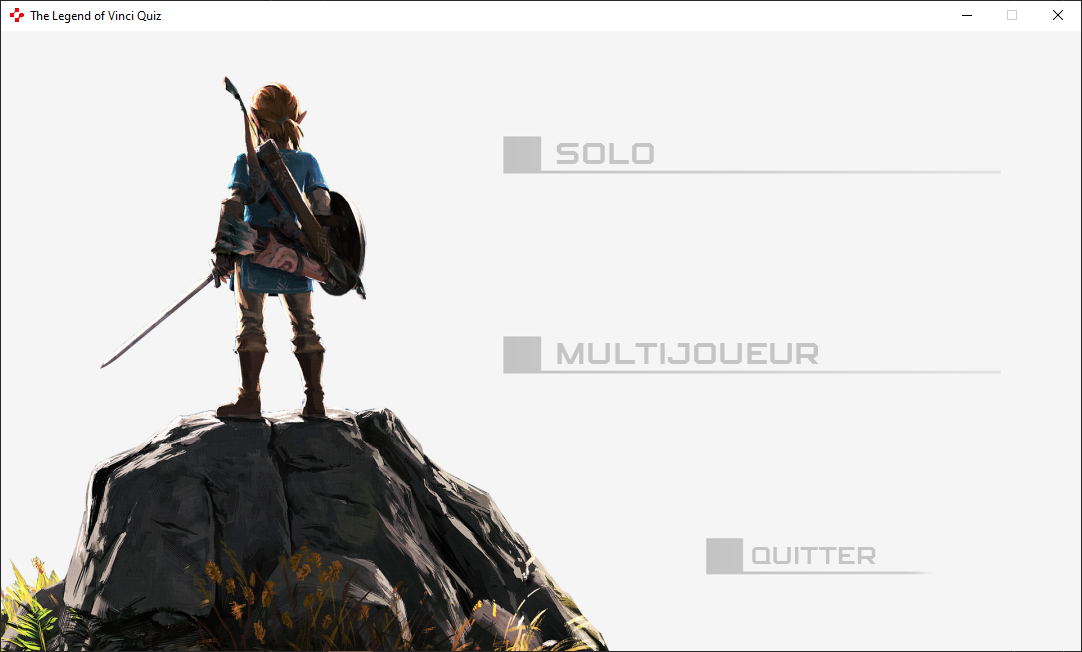


Figure : JPanel permettant de choisir le mode de jeu

Ce changement s’effectue à l’aide de la méthode suivante :

public void NextPanel(Object object) {

// Dans le cas où le JPanel est celui de connexion

if (object instanceof PnlLogin) {

laConsole.getPnlLogin().setVisible(false);

laConsole.remove(laConsole.getPnlLogin());

laConsole.setPnlLogin(null);

laConsole.setPnlGameMode(new PnlGameMode(this));

laConsole.getContentPane().add(laConsole.getPnlGameMode());

}

[…]

}

Cette méthode est utilisée si les informations saisies par l’utilisateur sont correctes :

private void login() {

try {

if(monController.verification(txtName.getText(), String.valueOf(passwordField.getPassword()))) {

// Changement de Panel

monController.NextPanel(monController.getLaConsole().getPnlLogin());

} else {

monController.getLaConsole().setBackground("img/PnlLogin/erreur\_login2.png");

monController.getLaConsole().remove(monController.getLaConsole().getPnlLogin());

monController.getLaConsole().add(monController.getLaConsole().getPnlLogin());

passwordField.setText("");

passwordField.requestFocus();

}

} catch (Exception e1) {

e1.printStackTrace();

}

}

Cette méthode est donc utilisée pour chaque JPanel ayant un JPanel à afficher après avoir effectué une action.

### Les boutons pour valider les saisies[[9]](#footnote-9)

Gérer les interactions avec un JButton impose d'utiliser le modèle Observer-Observable de Swing avec l'implémentation d'un listener ce qui sous-entend également de comprendre ce qu'est une interface au sens Java du terme[[10]](#footnote-10).

La notion d'interface est absolument centrale en Java, et massivement utilisée dans le design des API du JDK (cf. la vidéo de Langlet, Étienne - 10. Java, Interfaces). En Java, une interface est une sorte de classe qui spécifie et "contractualise" un comportement que les classes filles doivent mettre en œuvre. En cela, elles ressemblent aux protocoles réseaux. Les interfaces sont déclarées en utilisant le mot-clé interface et ne peuvent contenir que la signature des méthodes et les déclarations des constantes (variables qui sont déclarées à la fois statique et finale).

Java et Swing offrent plusieurs possibilités pour réaliser une interaction avec un composant, un JButton par exemple. Cette diversité nécessite de comprendre les concepts de classe interne, classe locale (interne de méthode) et classe anonyme[[11]](#footnote-11).

On peut directement à partir de la classe signer un contrat avec une interface en codant par exemple :

public class LaClasse extend JFrame **implements ActionListener** { […] }

On peut déclarer une classe interne qui implémentera l'interface ad hoc. Une classe interne a accès aux méthodes et attributs de la classe englobante.

Par exemple :

public class TestBouton extends JFrame {

JPanel panel\_1 = new JPanel();

JPanel panel\_2 = new JPanel();

public TestBouton() {

getContentPane().setLayout(null);

panel\_1.setBorder(new TitledBorder(null, "Commande", TitledBorder.LEADING, TitledBorder.TOP, null, null));

panel\_1.setBounds(10, 11, 422, 117);

getContentPane().add(panel\_1);

panel\_1.setLayout(null);

JButton btnStop = new JButton("Stop");

btnStop.setBounds(323, 83, 89, 23);

panel\_1.add(btnStop);

btnStop.**addActionListener(new changeColor())**;

[…]

}

**class changeColor implements ActionListener {**

**public void actionPerformed(ActionEvent e)**

**{**

**panel\_2.setBackground(Color.red);**

**System.out.println("Stop !");**

**}**

**}**

[…] }

Enfin, on peut utiliser des classes anonymes. Par exemple :

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

/\*\*

\* Classe englobante

\*/

public class ClasseEnglobante{

/\*\*

\* Méthode englobant l'appel à une classe anonyme

\*/

public void methodeEnglobante(){

/\*\*

\* Déclaration et instanciation de la classe anonyme pour un bouton

\* Le bouton est déclaré 'final' afin que la classe anonyme puisse y accéder

\*/

final JButton bouton = new JButton("monBouton");

bouton.addActionListener(**new ActionListener(){**

**public void actionPerformed(ActionEvent e){**

**System.out.println(bouton.toString());**

**}**

**}**);

}

}

#### Les boutons dans l’application

Dans l’application, chaque bouton doit avoir un background différent lorsque le joueur passe dessus avec la souris. Une classe nommée ButtonDisplay a donc été créée qui étend JButton.

public class ButtonDisplay extends JButton {

private ImageIcon backgroundImage;

public ButtonDisplay(int x, int y, int width, int height, String a, String b) {

setBounds(x, y, width, height);

InputStream input = getClass().getClassLoader().getResourceAsStream(a);

try {

backgroundImage = new ImageIcon(ImageIO.read(input));

} catch (IOException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

// backgroundImage = new ImageIcon(getClass().getClassLoader().getResource(a));

Image img = backgroundImage.getImage().getScaledInstance(getWidth(), getHeight(), Image.SCALE\_SMOOTH);

backgroundImage = new ImageIcon(img);

setIcon(backgroundImage);

// Supprimer la bordure du bouton

setBorderPainted(false);

[…]

}

Dans cette classe, une image de fond est affichée lorsque le joueur n’a pas la souris dessus, et une autre image est affichée lorsque le joueur a la souris dessus. Pour cela, il faut utiliser les méthodes mouseEntered et mouseExited qui permettent de capter quand la souris rentre sur le bouton et en sort.

[…]

// Ajouter un MouseListener pour changer l'image lorsqu'une souris passe dessus

addMouseListener(new MouseAdapter() {

public void mouseEntered(MouseEvent e) {

// Changer l'image de fond lorsque la souris rentre dans le bouton

InputStream input = getClass().getClassLoader().getResourceAsStream(b);

try {

ImageIcon background = new ImageIcon(ImageIO.read(input));

Image img = background.getImage().getScaledInstance(getWidth(), getHeight(), Image.SCALE\_SMOOTH);

background = new ImageIcon(img);

setIcon(background);

setCursor(new Cursor(Cursor.HAND\_CURSOR));

} catch (IOException e1) {

// TODO Auto-generated catch block

e1.printStackTrace();

}

}

public void mouseExited(MouseEvent e) {

// Changer l'image de fond lorsque la souris sort du bouton

setIcon(backgroundImage);

setCursor(new Cursor(Cursor.HAND\_CURSOR));

}

});

[…]

Pour ajouter un bouton, il faut donc instancier un ButtonDisplay en passant comme paramètre, ses dimension, ses cordonnes ainsi que les chemins des deux images devant être affichées.

ButtonDisplay btnLogin = new ButtonDisplay(50, 450, 250, 50, "img/PnlLogin/connexion\_eteint.png",

"img/PnlLogin/connexion\_allume.png");

Le résultat du bouton est présenté ci-dessous :

Une image contenant logo

Description générée automatiquement

Figure : Affichage d'un bouton lorsque la souris n'est pas dessus

Une image contenant texte, logo

Description générée automatiquement

Figure : Affichage d'un bouton lorsque la souris passe dessus

### Les boutons radio pour la sélection de la réponse[[12]](#footnote-12)

L'utilisateur peut choisir la réponse à une question grâce à quatre boutons radio disposés dans le JPanel qui regroupe toutes les réponses à une question.

Déclaration du JPanel et des boutons radio :

// Définit le JPanel de l'affichage des questions

pnlDisplayQuiz = **new** JPanel();

pnlDisplayQuiz.setBounds(10, 10, 678, 453);

pnlDisplayQuiz.setLayout(**null**);

pnlDisplayQuiz.setVisible(**false**);

pane.add(pnlDisplayQuiz);

pnlDisplayQuiz.setLayout(**null**);

JRadioButton rdbtn1 = **new** JRadioButton("");

rdbtn1.setActionCommand("0");

rdbtn1.setBounds(62, 167, 418, 21);

pnlDisplayQuiz.add(rdbtn1);

JRadioButton rdbtn2 = **new** JRadioButton("");

rdbtn2.setActionCommand("1");

rdbtn2.setBounds(62, 209, 418, 21);

pnlDisplayQuiz.add(rdbtn2);

JRadioButton rdbtn3 = **new** JRadioButton("");

rdbtn3.setActionCommand("2");

rdbtn3.setBounds(62, 246, 418, 21);

pnlDisplayQuiz.add(rdbtn3);

JRadioButton rdbtn4 = **new** JRadioButton("");

rdbtn4.setActionCommand("3");

rdbtn4.setBounds(62, 292, 418, 21);

pnlDisplayQuiz.add(rdbtn4);

JRadioButton rdbtn5 = **new** JRadioButton("");

rdbtn5.setActionCommand("4");

rdbtn5.setBounds(153, 340, 249, 21);

rdbtn5.setVisible(**false**);

rdbtn5.setSelected(**true**);

pnlDisplayQuiz.add(rdbtn5);

Les boutons radio peuvent être regroupé dans un groupe de bouton :

[…]

**private** ButtonGroup bgAnswer = **new** ButtonGroup();

[…]

bgAnswer.add(rdbtn1);

bgAnswer.add(rdbtn2);

bgAnswer.add(rdbtn3);

bgAnswer.add(rdbtn4);

bgAnswer.add(rdbtn5);

[…]

Lors de la validation d’une réponse, on récupère la valeur du bouton radio sélectionné afin de voir si la réponse à la question est juste ou fausse :

**private** **void** questionTreatment(ActionEvent event) {

// Num du bouton radio sélectionné

**int** bgSelected = Integer.*parseInt*(bgAnswer.getSelection().getActionCommand());

**if** (**this**.isValidAnswer(bgSelected)) {

// Changement de panel

pnlDisplayQuiz.setVisible(**false**);

pnlResultAnswer.setVisible(**true**);

lblErrorDisplayQuiz.setText("");

**if** (monController.getLaGame().isCorrectThisAnswer(currentQuestion, bgSelected)) {

// Affiche que la réponse est correcte

lblAnswer.setText("Bonne réponse Vous avez gagné 10 points");

} **else** {

// Affiche que la réponse est fausse

lblAnswer.setText("Mauvaise réponse");

}

} **else** {

lblErrorDisplayQuiz.setText("Veuillez séléctionner une réponse");

}

}

### Les listes déroulantes[[13]](#footnote-13)

En Swing, il existe 2 sortes de listes déroulantes :

1. JList, liste déroulante qui permet d'afficher et sélectionner plusieurs éléments à la fois.
2. JComboBox, liste de choix.

Il existe 2 manières de manipuler des JComboBox, soit on utilise directement les méthodes de manipulations des éléments de la JComboBox soit on développe son propre modèle de liste.

Dans l'application, une liste déroulante est utilisée afin de sélectionner le nombre de question pour le quiz. Cette liste est alimentée dynamiquement par groupe de 5 questions en fonction du nombre total de question présent dans la base de données. La méthode "addItem" permet de peupler la liste. De plus, la liste de l’application est créée à l’aide d’une méthode, renvoyant une "JCombox<Object>" :

[…]

**private** JComboBox<Object> createJComboBoxSolo()

**throws** FileNotFoundException, ClassNotFoundException, IOException, SQLException {

listeNbQuestionSolo = **new** JComboBox<>();

**int** nombreTotalQuestion = monController.getLaBase().nombreTotalQuestion();

nombreTotalQuestion = (**int**) (Math.*floor*(nombreTotalQuestion / 5));

**for** (**int** i = 1; i < nombreTotalQuestion + 1; i++) {

listeNbQuestionSolo.addItem(i \* 5);

}

**return** listeNbQuestionSolo;

}

[…]

Appelle de la méthode :

[…]

// Création de la liste déroulante pour le nombre de question

listeNbQuestionSolo = createJComboBoxSolo();

listeNbQuestionSolo.setFont(**new** Font("Tahoma", Font.***PLAIN***, 15));

listeNbQuestionSolo.setBounds(204, 265, 153, 48);

pnlSoloCreateGame.add(listeNbQuestionSolo);

[…]

Le résultat de cette liste est présenté ci-dessous :

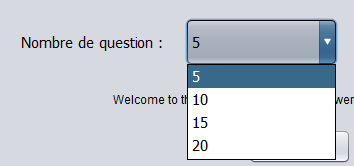


Figure : liste déroulante déterminant le nombre de question du quiz

# Cryptographie[[14]](#footnote-14)

## Différence entre Hachage et Cryptage[[15]](#footnote-15)



Figure : algorithme de Hachage

Un hachage peut simplement être défini comme un nombre généré à partir d’une chaîne de texte. En substance, un hachage est plus petit que le texte qui le produit. Il est généré de manière qu’un hachage similaire de même valeur ne puisse pas être produit par un autre texte. De cette définition, on peut voir que le hachage est le processus de production de valeurs de hachage dans le but d’accéder aux données et pour des raisons de sécurité dans les systèmes de communication. En principe, le hachage prendra une entrée arbitraire et produira une chaîne de longueur fixe. En règle générale, le hachage aura les attributs suivants :

* Une entrée donnée qui est connue, doit toujours produire une sortie connue.
* Une fois que le hachage a été fait, il devrait être impossible de passer de la sortie à l’entrée.
* Différentes entrées multiples devraient donner une sortie différente.
* Modifier une entrée devrait signifier un changement dans le hachage.

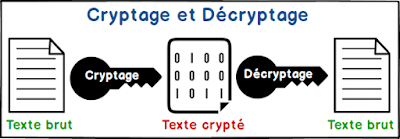


Figure : définition du cryptage

Le cryptage est le processus consistant à modifier des données d’une sorte de code secret afin de minimiser les risques d’accès aux données par des indiscrétions. C’est le moyen le plus efficace d’assurer la sécurité des données dans les systèmes de communication modernes. Pour que le destinataire puisse lire un message crypté, il doit avoir un mot de passe ou une clé de sécurité utilisée pour le décryptage. Les données qui n’ont pas été chiffrées sont connues sous le nom de texte brut tandis que les données chiffrées sont connues sous le nom de texte chiffré. Il existe un certain nombre de systèmes de cryptage, où un cryptage asymétrique est également connu sous le nom de cryptage à clé publique, le cryptage symétrique et le cryptage hybride sont les plus communs.

* Cryptage symétrique : Utilise la même clé secrète pour chiffrer et déchiffrer le message. La clé secrète peut être un mot, un nombre ou une chaîne de lettres aléatoires. L’expéditeur et le destinataire doivent tous deux avoir la clé. C’est la plus ancienne technique de cryptage.
* Cryptage asymétrique : Il déploie deux clés, une clé publique connue par tous et une clé privée connue uniquement par le récepteur. La clé publique est utilisée pour chiffrer le message et une clé privée est utilisée pour le déchiffrer. Le cryptage asymétrique est un peu plus lent que le cryptage symétrique et consomme plus de puissance de traitement lors du cryptage des données.

## Hachage : jBCrypt[[16]](#footnote-16)

jBCrypt implémente le hachage de mot de passe Blowfish de style OpenBSD en utilisant le schéma décrit dans "A Future-Adaptable Password Scheme"[[17]](#footnote-17) par Niels Provos et David Mazieres.

Ce système de hachage de mot de passe tente de contrecarrer le piratage de mot de passe hors ligne à l'aide d'un algorithme de hachage à forte intensité de calcul, basé sur le chiffrement Blowfish de Bruce Schneier. Le facteur de travail de l'algorithme est paramétrable, il peut donc être augmenté à mesure que les ordinateurs deviennent plus rapides.

De plus, jBCrypt utilise un sel pour hacher. Le salage est une méthode permettant de renforcer la sécurité des informations qui sont destinées à être hachées (par exemple des mots de passe) en y ajoutant une donnée supplémentaire afin d’empêcher que deux informations identiques ne conduisent à la même empreinte (la résultante d’une fonction de hachage). Le but du salage est de lutter contre les attaques par analyse fréquentielle, les attaques utilisant des rainbow tables, les attaques par dictionnaire et les attaques par force brute.

Pour hacher un mot de passe pour la première fois, il faut appeler la méthode hashpw() avec un sel aléatoire, comme ceci :

String pw\_hash = BCrypt.hashpw(plain\_password, BCrypt.gensalt());

Pour vérifier si un mot de passe en clair correspond à celui qui a été précédemment haché, il faut utiliser la méthode checkpw() :

if (BCrypt.checkpw(password, hashed\_password))

System.out.println("It matches");

else

System.out.println("It does not match");

### Implementation

Pour implémenter jBCrypt dans le projet, il faut ajouter le fichier BCrypt.java que l’on retrouve dans le zip téléchargeable sur le lien suivant :

<https://www.mindrot.org/projects/jBCrypt/>

Par défaut, BCrypt est une classe statique, avec des méthodes également statiques. Il n’est donc pas nécessaire d’instancier un objet BCrypt pour l’utiliser. Cependant, pour des raisons d’ergonomie de code, BCrypt n’est plus une classe statique. Il faut donc instancier un objet BCrypt afin de pouvoir se servir du hacheur :

BCrypt myHash = **new** BCrypt();

#### Vérification d’un mot de passe

Le hacheur est utilisé dans l’application afin de vérifier si le mot de passe saisi par l’utilisateur lors de l’identification correspond à son mot de passe haché enregistré dans la base de données. Pour cela, on utilise la méthode verifLogin() de la classe MySQLAccess qui vérifie la cohérence en les informations saisies par l’utilisateur et les informations de la base de données. Elle renvoie l’identifiant de l’utilisateur si les informations sont correctes, 0 sinon :

public int verifLogin(String name, String password) {

String query = "select \* from player where name\_player = ?";

try (Connection connection = DriverManager.getConnection(urlCnx, loginCnx, passwordCnx);

PreparedStatement ps = connection.prepareStatement(query)) {

ps.setString(1, name);

ps.execute();

ResultSet resultSet = ps.getResultSet();

if (resultSet.next()) {

String password\_db = resultSet.getString("password\_player");

BCrypt myHash = new BCrypt();

if (myHash.checkpw(password, password\_db)) {

int id\_db = resultSet.getInt("id\_player");

return id\_db;

}

return 0;

}

return 0;

} catch (Exception e) {

return 0;

}

}

## Chiffrement – déchiffrement : Jasypt[[18]](#footnote-18)

Jasypt est une bibliothèque Java qui permet au développeur d'ajouter des capacités de cryptage de base à ses projets avec un minimum d'effort et sans avoir besoin d'avoir une connaissance approfondie du fonctionnement de la cryptographie.

Il est possible de crypter les données avec une clé. Ensuite, il est possible de les déchiffrer avec la même clé.

### Implémentation

Pour effectuer le chiffrement et le déchiffrement à l'aide d'un algorithme très simple, nous pouvons utiliser une classe BasicTextEncryptor de la bibliothèque Jasypt, est créer une méthode permettant de crypter les données et de les retourner, le tout dans une classe nommée Jasypt.java, dans le package controller :

/\*\*

\* Crypte les données

\*

\* **@author** Nathan Ponson

\* **@param** text Les données à crypter

\* **@return** Les données cryptées

\*/

**public** String encrypt(String text) {

BasicTextEncryptor textEncryptor = **new** BasicTextEncryptor();

textEncryptor.setPasswordCharArray(password);

String encryptedText = textEncryptor.encrypt(text);

**return** encryptedText;

}

De ce fait, nous pouvons également créer une méthode de déchiffrement des données en envoyant les données chiffrées, et qui retourne les données déchiffrées :

/\*\*

\* Décrypte les données

\*

\* **@author** Nathan Ponson

\* **@param** text Les données à décrypter

\* **@return** Les données décryptées

\*/

**public** String decrypt(String text) {

BasicTextEncryptor textEncryptor = **new** BasicTextEncryptor();

textEncryptor.setPasswordCharArray(password);

String decryptedText = textEncryptor.decrypt(text);

**return** decryptedText;

}

Pour pouvoir utiliser ces méthodes, il faut créer un objet theDecrypter dans le contrôleur :

/\*\*

\* Décrypteur de l'application

\*/

**private** Jasypt theDecrypter;

[…]

**public** Controller(){

[…]

// Initialise le décrypteur de l'application

**this**.theDecrypter = **new** Jasypt();

[…]

}

Il faut donc un constructeur dans la classe Jasypt.java permettant de créer l’objet.

/\*\*

\* Crée le décrypteur

\*

\* **@author** Nathan Ponson

\*/

**public** Jasypt (Controller aController) {

//Mot de passe permettant de chiffrer et déchiffrer

password = "4C45Lrh26YWbKz73FwtXYkc6".toCharArray();

}

Le déchiffrement est utilisé dans l’application afin de déchiffrer les données de connexion à la base de données :

// Charge le driver

try {

Class.forName(monController.getTheDecrypter().decrypt(properties.getProperty("jdbc.driver.class")));

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

// Initialise les paramètres de connexions

urlCnx = monController.getTheDecrypter().decrypt(properties.getProperty("jdbc.url"));

loginCnx = monController.getTheDecrypter().decrypt(properties.getProperty("jdbc.login"));

passwordCnx = monController.getTheDecrypter().decrypt(properties.getProperty("jdbc.password"));

# Jeu multijoueur : serveur websocket

Un serveur WebSocket est un serveur informatique capable de communiquer avec les clients via une connexion WebSocket, qui est une technologie de communication en temps réel bidirectionnelle entre un client et un serveur.

Contrairement aux protocoles HTTP traditionnels qui sont basés sur une communication client-serveur par requêtes et réponses, les connexions WebSocket sont établies une seule fois et maintenues ouvertes pour permettre une communication en temps réel entre le client et le serveur. Cela permet aux serveurs WebSocket de transmettre rapidement des données à un grand nombre de clients en temps réel, tels que des messages de chat, des mises à jour en temps réel de jeux, des informations de trading en temps réel, etc.

Les serveurs WebSocket sont souvent utilisés pour les applications web qui développent une communication en temps réel avec les clients, comme les jeux en ligne, les réseaux sociaux, les applications de trading, les applications de suivi en temps réel, les applications de chat et de messagerie, etc...

## KryoNet

KryoNet est une bibliothèque Java open source pour la communication basée sur TCP et UDP, et qui prend également en charge la communication WebSocket. KryoNet utilise la sérialisation Kryo pour transférer les objets Java via le réseau, ce qui permet une communication efficace entre les clients et le serveur.

KryoNet facilite la création de serveurs et de clients pour les jeux en réseau, les applications de messagerie, les applications de suivi en temps réel, et d'autres applications qui accèdent à une communication réseau en temps réel.

Le code ci-dessous montre comment établir une connexion côté client :

public class WebSocketClientExample {

public static void main(String[] args) throws IOException {

// Crée un nouveau client KryoNet

Client client = new Client();

// Enregistre les classes que le client doit sérialiser / désérialiser

client.getKryo().register(MyObject.class);

// Connecte le client à l'adresse IP et au port du serveur WebSocket

client.start();

client.connect(5000, InetAddress.getByName("localhost"), 8080);

// Ajoute un écouteur de connexion pour gérer les événements de connexion, déconnexion et les données reçues

client.addListener(new Listener() {

public void connected(Connection connection) {

System.out.println("Connexion établie");

}

public void disconnected(Connection connection) {

System.out.println("Connexion fermée");

}

public void received(Connection connection, Object object) {

if (object instanceof MyObject) {

MyObject myObject = (MyObject) object;

System.out.println("Objet reçu: " + myObject.toString());

}

}

});

// Envoie un objet au serveur WebSocket

MyObject myObject = new MyObject();

myObject.data = "Hello World!";

client.sendTCP(myObject);

// Attend une seconde avant de fermer la connexion et de terminer l'exécution du programme

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

client.stop();

}

// Classe à sérialiser / désérialiser

public static class MyObject {

public String data;

public String toString() {

return data;

}

}

}

## Implémentation

### Connexion / Déconnexion

Dans notre application, nous avons une classe nommée ClientWebsocket permettant d’établir la connexion avec le serveur websocket.

public class ClientWebsocket {

private Client client = new Client(10000000, 10000000);

private Controller monController;

public ClientWebsocket(Controller unController) throws IOException {

monController = unController;

Kryo kryo = client.getKryo();

kryo.register(SomeRequest.class);

kryo.register(Message.class);

kryo.register(ArrayList.class);

kryo.register(QuizGame.class);

kryo.register(Answer.class);

kryo.register(Player.class);

kryo.register(Question.class);

kryo.register(Game.class);

kryo.register(LesGame.class);

client.start();

client.connect(5000, "127.0.0.1", 54556);

[…]

}

}

La connexion est créée au moment où le jouer choisi le mode de jeu et choisi le mode multijoueur.

[…]

if (isMulti()) {

try {

setLeClient(new ClientWebsocket(this));

laConsole.setPnlMultiGameMode(new PnlMultiGameMode(this));

laConsole.getContentPane().add(laConsole.getPnlMultiGameMode());

} catch (IOException e) {

// e.printStackTrace();

laConsole.setPnlGameMode(new PnlGameMode(this));

laConsole.getContentPane().add(laConsole.getPnlGameMode());

JOptionPane.showMessageDialog(laConsole,

"Un problème est survenue lors de la connexion au serveur.\r\nVeuillez réessayer.",

"Erreur de connexion", JOptionPane.ERROR\_MESSAGE);

}

}

[…]

La connexion est par la suite supprimée au moment où le joueur qui le mode multijoueur.

[…]

setMulti(false);

leClient.getClient().close();

[…]

### Envoi / Réception d’un message

Pour communiquer avec le serveur, une classe nommé Message a été créée. Elle contient un integer option qui permet de réaliser un traitement en fonction de l’option. Elle contient également d’autre objet permettant le transfert de données lors d’un envoi de message.

public class Message {

private int option;

private int idGame;

private Player player;

private ArrayList<Player> lesPlayer;

public Message() {

}

[…]

}

Lors d’un envoi de message, on instancie un objet Message en mettant un chiffre décrivant le traitement à réaliser.

public void joinGame(int idGame) {

client.sendTCP(new Message(1, idGame, monController.getMonPlayer()));

}

Lors d’une réception de message, nous récupérons l’option, et en fonction, nous choisissons une fonction à effectuer.

public void selectOption(Message message) {

if (message.getOption() == 2) { // Start game

startGame();

} else if (message.getOption() == 3) { // Impossible de rejoindre la game

joinGameInpossible();

} else if (message.getOption() == 4) { // Mise à jour des scores

setListPlayerGame(message);

}

}

# JAR - archivage - déploiement et signature[[19]](#footnote-19)

## Le package JAR archive

Il s'agit d'une archive contenant des fichiers Java compilés (des fichiers .class). Le terme de Jar signifie Java ARchive. Cette archive est au format ZIP : vous pouvez donc ouvrir un fichier JAR avec n'importe quel outil de manipulation de fichier ZIP (7-Zip, WinZip, Gestionnaire d'archives Linux...).

Il est possible de créer une archive JAR avec un outil de compression ZIP quelconque. Pour autant il est vivement conseillé d'utiliser l'outil jar[.exe] proposé par votre JDK (Java Development Kit). Effectivement, outre les fichiers .class, un JAR doit contenir un fichier de « manifeste » : il doit être obligatoirement se nommer META-INF/MANIFEST.MF. L'outil jar gère de manière automatisée la production de ce fichier de manifeste.

### Génération d’un jar file assistée par Eclipse

Il est possible de produire un fichier Jar à partir d'Eclipse. Pour ce faire, il faut cliquer avec le bouton droit de la souris sur le projet, cliquer sur « Export... », puis sélectionner l'assistant « JAR file », comme montré dans la capture d'écran ci-dessous.

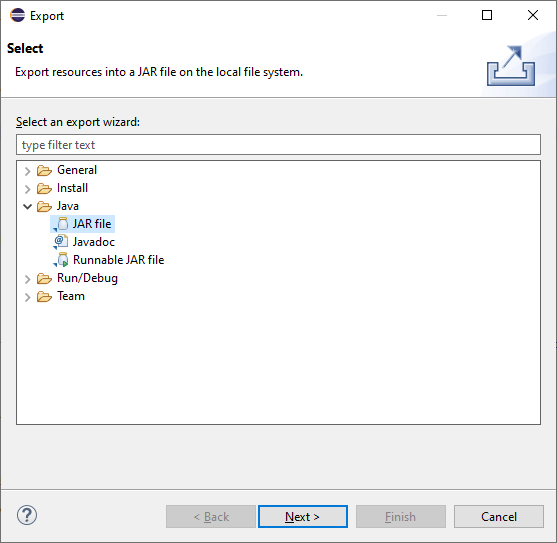


Figure : génération d'un jar file à l'aide d'éclipse - étape 1

Il ne reste plus qu'à sélectionner les éléments à archiver et à spécifier le nom du fichier Jar. Voici une capture d'écran de cet assistant :

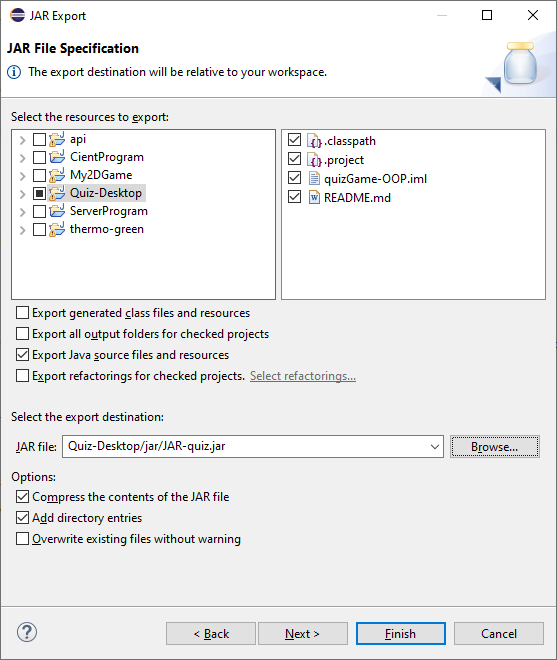


Figure : génération d'un jar file à l'aide d'éclipse - étape 2

## Le package JAR runnable

L'outil jar (Java Archive) du JDK permet également de créer un fichier jar exécutable. Un fichier jar exécutable appelle la méthode principale de la classe lors d’un double-clique dessus.

### Génération d’un jar file assistée par Eclipse

Il est possible de produire un fichier Jar exécutable à partir d'Eclipse. Pour ce faire, il faut cliquer avec le bouton droit de la souris sur le projet, cliquer sur « Export... », puis sélectionner l'assistant « JAR runnable », comme montré dans la capture d'écran ci-dessous.

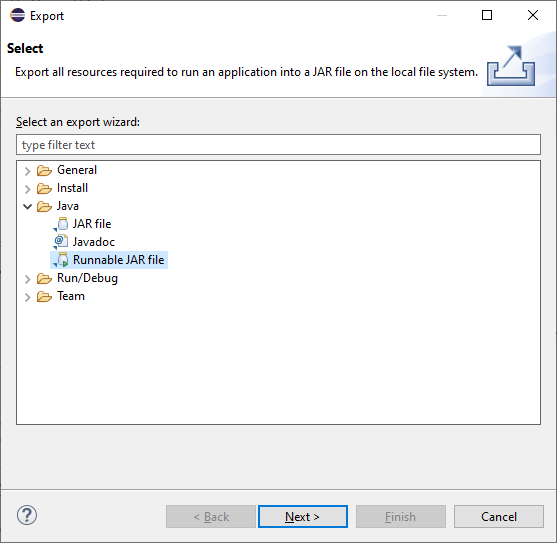


Figure : génération d'un jar runnable à l'aide d'éclipse - étape 1

Il faut par la suite sélectionner les éléments à archiver et à spécifier le nom du fichier Jar ainsi que sélectionner la configuration de lancement de l’application. Voici une capture d'écran de cet assistant :

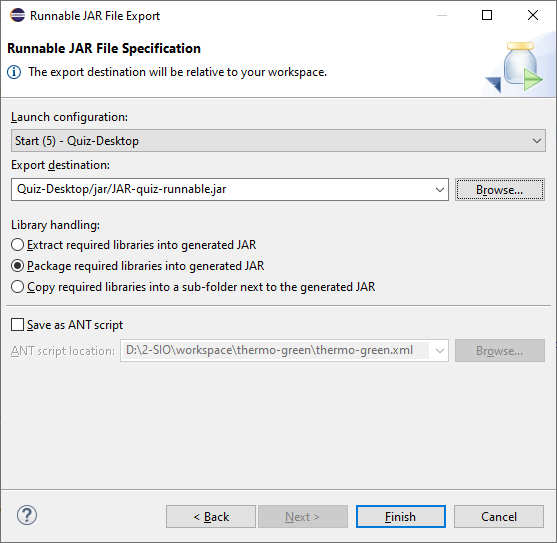


Figure : génération d'un jar runnable à l'aide d'éclipse - étape 2

# TABLE DES ILLUSTRATIONS

[Figure 1 : principe de fonctionnement du modèle MVC 2](#_Toc133046415)

[Figure 2 : diagramme des composants de l’application 3](#_Toc133046416)

[Figure 3 : dossiers essentiels du Build 4](#_Toc133046417)

[Figure 4 : diagramme des classes métiers 4](#_Toc133046418)

[Figure 5 : principe de fonctionnement de l'API JDBC 6](#_Toc133046419)

[Figure 6 : les différentes méthodes de connexion avec JDBC 7](#_Toc133046420)

[Figure 7 : JPanel permettant de se connecter 12](#_Toc133046421)

[Figure 8 : JPanel permettant de choisir le mode de jeu 12](#_Toc133046422)

[Figure 9 : Affichage d'un bouton lorsque la souris n'est pas dessus 16](#_Toc133046423)

[Figure 10 : Affichage d'un bouton lorsque la souris passe dessus 16](#_Toc133046424)

[Figure 11 : liste déroulante déterminant le nombre de question du quiz 18](#_Toc133046425)

[Figure 12 : algorithme de Hachage 19](#_Toc133046426)

[Figure 13 : définition du cryptage 19](#_Toc133046427)

[Figure 14 : génération d'un jar file à l'aide d'éclipse - étape 1 26](#_Toc133046428)

[Figure 15 : génération d'un jar file à l'aide d'éclipse - étape 2 27](#_Toc133046429)

[Figure 16 : génération d'un jar runnable à l'aide d'éclipse - étape 1 27](#_Toc133046430)

[Figure 17 : génération d'un jar runnable à l'aide d'éclipse - étape 2 27](#_Toc133046431)

1. <http://www.infres.enst.fr/~hudry/coursJava/interSwing/boutons5.html> [↑](#footnote-ref-1)
2. <https://fr.wikipedia.org/wiki/Objet_d'acc%C3%A8s_aux_donn%C3%A9es> [↑](#footnote-ref-2)
3. <https://fr.wikipedia.org/wiki/.properties>

   <https://www.techiedelight.com/fr/read-properties-files-java/> [↑](#footnote-ref-3)
4. <https://devstory.net/10167/java-jdbc#a12649> [↑](#footnote-ref-4)
5. <https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-collections-d-objets>

   <http://www.jmdoudoux.fr/java/dej/chap-collections.html> [↑](#footnote-ref-5)
6. http://blog.lecacheur.com/2006/04/01/stringequals-ou-stringcompareto/

   http://thecodersbreakfast.net/index.php?post/2008/02/22/24-comparaison-des-chaines-accentuees-en-java

   http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/ChainesDeCaracteres.html [↑](#footnote-ref-6)
7. <http://icps.u-strasbg.fr/~bastoul/teaching/java/docs/Swing.pdf>

   <http://www.jmdoudoux.fr/java/dej/chap-swing.html>

   <http://codes-sources.commentcamarche.net/faq/360-swinguez-jframe-jpanel-jcomponent-layoutmanager-borderlayout> [↑](#footnote-ref-7)
8. http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html [↑](#footnote-ref-8)
9. http://java.developpez.com/faq/gui?page=Les-listeners

   http://codes-sources.commentcamarche.net/faq/369-swing-partie-2-actionlistener-listener-jbutton [↑](#footnote-ref-9)
10. https://en.wikipedia.org/wiki/Interface\_%28Java%29

    https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-classes-abstraites-et-les-interfaces

    https://fr.wikibooks.org/wiki/Programmation\_Java/Interfaces

    https://docs.oracle.com/javase/tutorial/java/concepts/interface.html

    https://docs.oracle.com/javase/tutorial/java/IandI/usinginterface.html

    http://blog.paumard.org/cours/java/chap07-heritage-interface-interface.html [↑](#footnote-ref-10)
11. https://fr.wikipedia.org/wiki/Classe\_interne

    https://fr.wikibooks.org/wiki/Programmation\_Java/Classes\_internes

    http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/classes3.html#locale [↑](#footnote-ref-11)
12. https://docs.oracle.com/javase/tutorial/uiswing/components/button.html#radiobutton [↑](#footnote-ref-12)
13. <https://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html>

    <http://baptiste-wicht.developpez.com/tutoriels/java/swing/debutant/?page=listes>

    <https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-champs-de-formulaire> [↑](#footnote-ref-13)
14. <https://www.cnil.fr/fr/comprendre-les-grands-principes-de-la-cryptologie-et-du-chiffrement> [↑](#footnote-ref-14)
15. <https://waytolearnx.com/2018/07/difference-entre-cryptage-et-hachage.html> [↑](#footnote-ref-15)
16. <https://www.javadoc.io/doc/org.mindrot/jbcrypt/0.4/org/mindrot/jbcrypt/BCrypt.html>

    <https://fr.wikipedia.org/wiki/Bcrypt> [↑](#footnote-ref-16)
17. <https://harrymoreno.com/assets/greatPapersInCompSci/A_Future-Adaptable_Password_Scheme_-_provos_(1999).pdf> [↑](#footnote-ref-17)
18. <http://www.jasypt.org/>

    <https://www.devglan.com/online-tools/jasypt-online-encryption-decryption> [↑](#footnote-ref-18)
19. <https://en.wikipedia.org/wiki/JAR_%28file_format%29>

    <https://koor.fr/Java/Tutorial/java_archive_jar.wp#create_command_line>

    <https://docs.oracle.com/javase/tutorial/deployment/jar/> [↑](#footnote-ref-19)