Spritebot-v3
Programmer-Handbook
v.1.0.0

PAGE DE SERVICE

Référence : Spritebot
Plan de classement : stadium-technic-analyse-conception-spritebot
Niveau de confidentialité : confidential

Mises à jour

Version	Date	Auteur	Description du changement
1.0.0	20-10-2023	Thibaud MARCQ	Introduction, Objectifs, Architecture du code

Validation

Version	Date	Nom	Rôle

Diffusion

Version	Date	Nom	Rôle

Sommaire

1 INTRODUCTION

2 OBJECTIFS

3 REALISATION DU CODE EN JAVA

La réalisation du code qui permettra de remplir les objectifs et de créer un jeu de quizz en application se fera tout d'abord en langage JAVA sur l'IDE d'Eclipse. On commencera par la structuration de notre programme avant de passer à l'écriture du code où l'on s'intéressera à la conception de frame, de connexion à une BDD et la protection lors de la connexion de l'utilisateur.

3.1 ARCHITECTURE DU PROGRAMME

Tout d'abord, on va structurer notre programme, afin qu'il soit plus lisible à lire et à comprendre. Pour cela, nous diviserons en plusieurs packages les différentes classes en fonction de leurs rôles dans le code. Nous verrons en quoi consiste ces packages et leurs classes associées. L'architecture utilisée est sous format MVC (Model-Vue-Controller) et regroupera donc principalement ces trois packages.

3.1.1 PACKAGE CONTROL

Le package Control est celui qui sera au cœur du code dans ce package, se trouvera les classes Start et Controller. L'une servant à instancier l'application et l'autre permettant de contrôler les classes provenant d'autres packages. On note aussi la présence de la classe Configuration qui aura pour but de crypter les données présentes dans le fichier de configuration.

3.1.2 PACKAGE MODEL

Le package Model quant à lui contiendra les classes Answer, Player, Question et QuizGame qui sont comme des entités que l'on trouvera dans la base de données. Ce sont ces classes qui font en sorte que ces entités existent dans le programme au niveau Java.

3.1.3 PACKAGE VIEW

Le package View contiendra toutes les classes faisant offices de frame ou d'interface graphique, on y retrouve les classes Login, ChangePassword, GameStart, QuizGameGUI et RecapGame. Le code de ces classes se traduira par ce que l'on retrouvera lors du lancement de l'application.

3.1.4 PACKAGE DAO

Le package DAO servira à établir la connexion avec la base de données, on y retrouve trois classes, DAOsqlAnswer, DAOsqlQuestion et DataFileUser. Toutes ces classes auront pour but de se connecter à la base de données afin de récupérer l'identifiant de l'utilisateur, les questions ainsi que leurs réponses.

3.1.5 PACKAGE TOOLS

Le package Tools ne contiendra que la classe BCrypt, une classe utilisée pour le cryptage de données, c'est ce qu'on utilisera pour sécuriser la connexion utilisateur afin d'empêcher toutes tentatives de vols de données.

3.2 PROGRAMME PRINCIPAL

Maintenant que nous avons une vision de l'architecture que prendra notre code, nous pouvons maintenant nous intéresser ce en quoi consistera le programme que nous coderons en Java. Comme dit précédemment, les classes seront créés afin de remplir un rôle, par exemple, la classe Questions va permettre l'instanciation de variables questions. Ou encore la classe DataFileUser qui permettra de vérifier si le login et le password de l'utilisateur sont correcte avant de pouvoir permettre la connexion à l'application.

3.2.1 CLASSE CONTROLLER

Le controller est une classe essentielle dans notre code car c'est lui qui va interagir avec toutes les classes sources. Il permet notamment la liaison entre les classes Model et View.

3.2.2 CLASSES MODEL

Pour les classes Model, on retrouvera le même schéma, c'est-à-dire, tout d'abord, une phase de spécifications des variables de classe. Si on prend l'exemple de la classe Answer, le code ressemble à l'image ci-dessous.

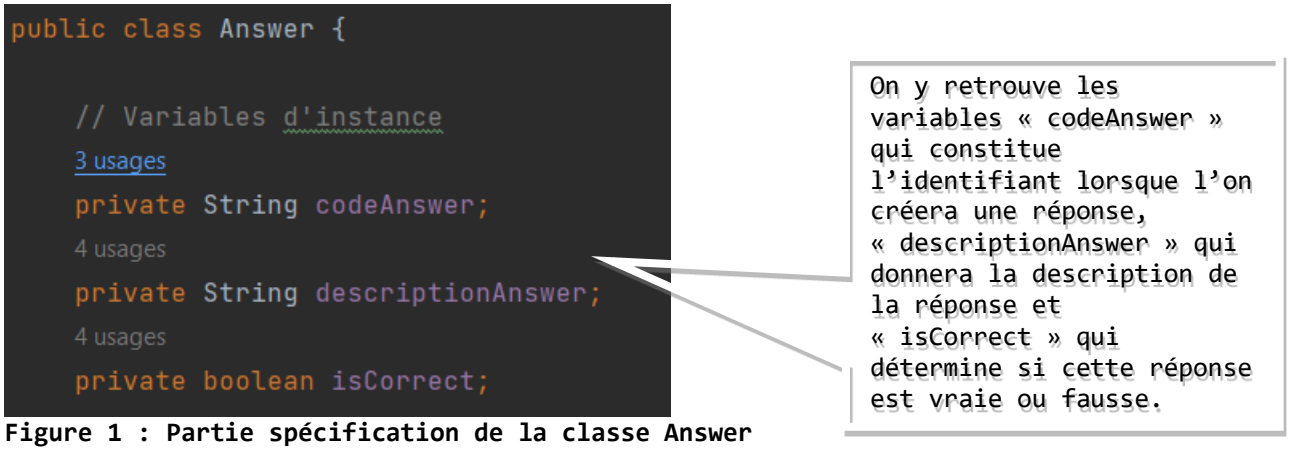

Figure 1 : Partie spécification de la classe Answer

S'ensuit la mise en place d'un constructeur, on y mettra en paramètres les variables « descriptionAnswer » et « isCorrect ».

```
// Constructeur
public Answer(String descriptionAnswer, boolean isCorrect) {
    this.descriptionAnswer = descriptionAnswer;
    this.isCorrect = isCorrect;
}
```

Figure 2 : Partie Implémentation (Constructeur) de la classe Answer

Puis pour terminer, on y ajoute aussi les getters et setters, qui nous serons utiles lorsque l'on voudra obtenir les variables à partir d'autres tables.

On fait un getter / setter par variables crées dans la classe, donc toutes les variables que l'on peut trouver dans la partie **spécifications**.

```java
// Getter pour la description de la réponse
public String getDescriptionAnswer() {
    return descriptionAnswer;
}

// Setter pour la description de la réponse
public void setDescriptionAnswer(String descript
    this.descriptionAnswer = descriptionAnswer;
}

// Getter pour savoir si la réponse est correcte
public boolean getIsCorrect() {
    return isCorrect;
}

// Setter pour définir si la réponse est correct
public void setCorrect(boolean correct) {
    isCorrect = correct;
}

// Getter pour le code de la réponse
public String getCodeAnswer() {
    return codeAnswer;
}

// Setter pour définir le code de la réponse
public void setCodeAnswer(String codeAnswer) {
    this.codeAnswer = codeAnswer;
}
```

Figure 3 : Partie Implémentation (Getter & Setter) de la classe Answer

Une fois ces trois bouts de codes réalisés, on peut délaisser les classes Model afin de se concentrer sur la partie graphique du code.

3.2.3 Classes Vues

Les classes que l'on mettra dans le package **View** sont celles qui permettront d'afficher l'interface graphique de l'application.

Tout comme les autres classes, il faut spécifier les variables, en différence que nous utilisons Swing et que cela nous demande de créer des variables comme **JPanel** ou **JButton.**

```java
public class GameStart extends JFrame {

    // Contrôleur associé à la vue
    private Controller myController;

    // Panneau de contenu
    private JPanel contentPane;
```

Figure 4 : Partie spécifications de la classe GameStart

On se sert ici de la classe Vue « GameStart » qui permet de lancer le jeu une fois sur la page principale du programme. On spécifie donc que nous avons besoins du Controller mais aussi de créer un Panel (ou JPanel) que l'on nommera « contentPane ».

S'ensuit donc la partie constructeur avec l'initialisation des variables spécifiées.

```
// Constructeur
public GameStart(Controller unController) {
    // Initialise la vue avec le contrôleur spécifié
    this.myController = unController;

    // Paramètres de la fenêtre
    setResizable(false);
    setIconImage(Toolkit.getDefaultToolkit().getImage("img\\sb-logo-monogram-circle.jpg"));
    setTitle("Sprite bot");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 200);

    // Panneau de contenu
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    // Bouton "Start"
    JButton btnStart = new JButton("Start");
    btnStart.setBounds(160, 82, 89, 23);
    contentPane.add(btnStart);

    // Ajout d'un écouteur d'événements pour le bouton "Start"
    btnStart.addActionListener(new ActionListener() {
```

Figure 5 : Partie implémentation de la classe GameStart

Ce qui change d'avec les autres classes, c'est que nous n'avons pas besoin de getters & setters mais qu'il faut paramétrez les éléments Swing. On peut voir sur l'image ci-dessus que nous paramétrons certaines valeurs avec la JFrame comme la taille, le titre, les bordures, le fait de pouvoir changer sa taille, l'ajout d'une icône et l'opération de fermeture par défaut.

En plus, nous avons le paramétrage du JPanel, puis l'ajout d'un bouton que l'on nomme « Start » et qui servira à lancer le jeu une fois cliqué, amenant à écrire un **ActionListener** dans notre code, comme celui que l'on voit en dessous.

```
public void actionPerformed(ActionEvent e) {
    // Appelle la méthode du contrôleur pour créer l'IHM du quiz
    myController.CreateQuizGameGUI();

    // Ferme la fenêtre actuelle
    dispose();

    // Affiche un message dans la console
    System.out.println("Game started!");
}
```

Figure 6 : Méthode actionPerformed

La méthide fait que l'on appelle le Controller à créer une IHM du quiz, ce qui ferme la fenetre actuelle et affiche le message « Game started ! ». Cette méthode permet de faire comprendre au joueur que le jeu est lancé.

3.3 Cas d'utilisation « Se Connecter »

Comme on l'a vu sur l'Analyse-Conception, nous devons permettre à l'utilisateur et donc joueur de pouvoir avoir son propre compte et donc de pouvoir se connecter à celui-ci.

Pour cela, nous créerons une classe Login dans le package View, on spécifiera les variables (Controller, JPanel et JTextField) afin que l'utilisateur puisse rentrer son identifiant et mot de passe.

```
package View;

import javax.swing.JFrame;

public class Login extends JFrame {

    // Contrôleur associé à la vue
    private Controller myController;

    // Panneau de contenu
    private JPanel contentPane;

    // Champs de texte pour le login et le mot de passe
    private JTextField txtLogin;
    private JTextField txtPwd;
```

Figure 7 : Partie spécification de la classe Login

Puis la partie implémentation va permettre de paramétrer les variables, on y mettra les labels pour login et password, les zones de textes où l'on rentre les informations et le bouton login afin de faire vérifier si les données entrer par l'utilisateur sont correcte.

```
// Labels pour le login et le mot de passe
JLabel lblLogin = new JLabel("Login : ");
lblLogin.setBounds(10, 11, 46, 14);
contentPane.add(lblLogin);

JLabel lblPwd = new JLabel("Password :");
lblPwd.setBounds(10, 36, 62, 14);
contentPane.add(lblPwd);

// Champs de texte pour le login et le mot de passe
txtLogin = new JTextField();
txtLogin.setBounds(87, 11, 86, 20);
contentPane.add(txtLogin);
txtLogin.setColumns(10);

txtPwd = new JPasswordField();
txtPwd.setColumns(10);
txtPwd.setBounds(87, 36, 86, 20);
contentPane.add(txtPwd);
```

On peut voir que les labels possèdent leurs bordures ainsi qu'un texte en plus d'être ajouté dans le « contentPane ».
Pour les JTextField, on paramétre le nombres de colonnes, les bordures et l'ajout dans le « contentPane ».

Figure 8 : Partie implémentation (JLabel & JTextField) de la classe Login

Sur le JButton, en oplus du paramètre, on y ajoute la méthode « actionPerformed » qui lui permettra de vérifier l'authentification et si les données inscrites par l'utilisateur sont correctes, notamment en utilisant le « .getText() » et que si les données sont correctes, cela change la fenetre et affiche la page **GameStart**.

```
// Bouton "Login"
JButton btnLogin = new JButton("Login");
btnLogin.setBounds(23, 82, 89, 23);
contentPane.add(btnLogin);

// Bouton "Change Password"
JButton btnChangePassword = new JButton("Change Password");
btnChangePassword.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Récupère le nom d'utilisateur et le mot de passe
        String nomUtilisateur = txtLogin.getText();
        String motDePasse = txtPwd.getText();

        // Vérifie l'authentification
        if (unController.verifyUserLogin(nomUtilisateur, motDePasse)) {
            // Affiche la fenêtre de changement de mot de passe
            myController.CreateFrameChangePassword(nomUtilisateur);
            dispose(); // Ferme la fenêtre actuelle
        } else {
            // Affiche un message d'erreur
            JOptionPane.showMessageDialog(null, "Nom d'utilisateur ou mot de passe incorrect.");
        }
    }
});
btnChangePassword.setBounds(134, 82, 118, 23);
contentPane.add(btnChangePassword);
```

Figure 9 : Partie implémentation (JButton & ActionListener) de la classe Login

```
// Ajout d'un écouteur d'événements pour le bouton "Login"
btnLogin.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Récupère le nom d'utilisateur et le mot de passe
        String nomUtilisateur = txtLogin.getText();
        String motDePasse = txtPwd.getText();

        // Vérifie l'authentification
        if (unController.verifyUserLogin(nomUtilisateur, motDePasse)) {
            // Affiche la fenêtre de démarrage du jeu
            myController.CreateGameStart();
            dispose(); // Ferme la fenêtre actuelle
        } else {
            // Affiche un message d'erreur
            JOptionPane.showMessageDialog(null, "Nom d'utilisateur ou mot de passe incorrect.");
        }
    }
});
```

Figure 10 : Partie implémentation ActionListener de la classe Login

3.4 CONNEXION A LA BASE DE DONNEES

Afin de pouvoir rentrer les informations tels que les données utilisateurs ainsi que les questions et réponses, nous allons créer une base de données. Cette base de données sera sous MySQL et portera le nom de « sbcg » soit SpriteBot Cybersecurity Game.

Pour cela, on va créer une classe **Configuration** ainsi qu'un fichier de configuration que l'on nomme « vtg.cfg », ce fichier comportera les données nécessaires à la connexion à la base de données.

La capture d'écran ci-dessous montre comment fonctionne la configuration avec le fichier ainsi que la lecture. On voit aussi que l'on paramètre un mot de passe par défaut « P@ssw0rdsio ».

```
// Constructeur
public Configuration() {
    // Initialise les propriétés et l'encrypteur
    this.properties = new Properties();
    this.encryptor = new StandardPBEStringEncryptor();

    // Définit le mot de passe pour l'encrypteur
    this.encryptor.setPassword("P@ssw0rdsio");

    try (FileInputStream input = new FileInputStream("./data/vtg.cfg")) {
        // Charge les propriétés de configuration à partir du fichier
        properties.load(input);
    } catch (IOException e) {
        // Gère une éventuelle IOException si le chargement du fichier échoue
        e.printStackTrace();
    }
}

// Méthode pour lire une propriété et la décrypter
public String readProperty(String theKey) {
    // Décrypte la valeur de la propriété en utilisant l'encrypteur
    return encryptor.decrypt(properties.getProperty(theKey));
}
```

Figure 11 : Configuration de la connexion à la base de données

Pour utiliser la base de données, on prend par exemple la classe **DAOsqlQuestion** qui sert à récupérer les questions inscrites dans la base de données. La classe ci-dessous fait appel à la configuration (le nom de la base de données, l'username et la mort de passe).

```
// Implémentation
public DAOsqlQuestion(Controller theController) {
    try {
        // Initialise le contrôleur et établit la connexion à la base de données
        this.myController = theController;
        String dbname = this.myController.getMyConfiguration().readProperty("database.url");
        String username = this.myController.getMyConfiguration().readProperty("database.username");
        String password = this.myController.getMyConfiguration().readProperty("database.password");

        this.connection = DriverManager.getConnection(dbname, username, password);

        this.statement = connection.createStatement();
    } catch (SQLException e) {
        // Gère les exceptions liées à la connexion à la base de données
        e.printStackTrace();
    }
}
```

Figure 12 : Obtention des données de la base de données (DAOsqlQuestion)

4 REALISATION DU CODE SOUS ANDROID STUDIO

5 CONCLUSION

6 INDEX DES ILLUSTRATIONS