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	Julian CUCHERAT	Création Networking Inc.
1.1.0	10-12-2022	Nathan PONSON	MAJ

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

1 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] :

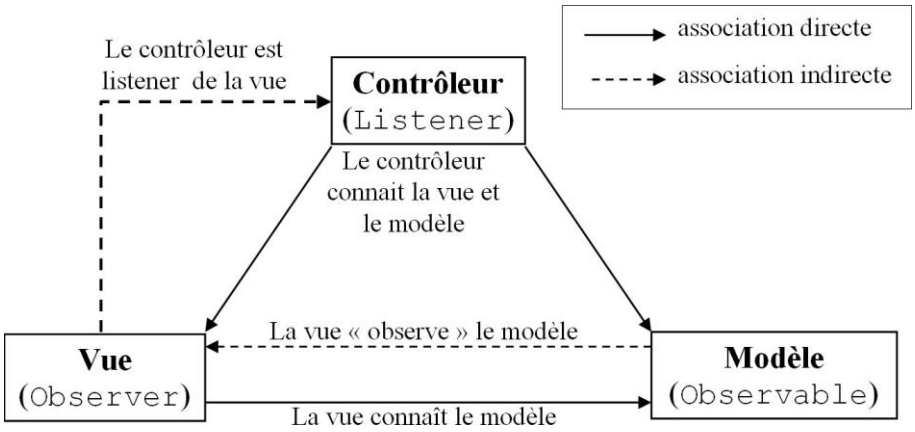


Figure 1 : principe de fonctionnement du modèle MVC

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] 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 :

[1] http://www.infres.enst.fr/~hudry/coursJava/interSwing/boutons5.html
[2] https://fr.wikipedia.org/wiki/Objet_d'acc%C3%A8s_aux_donn%C3%A9es

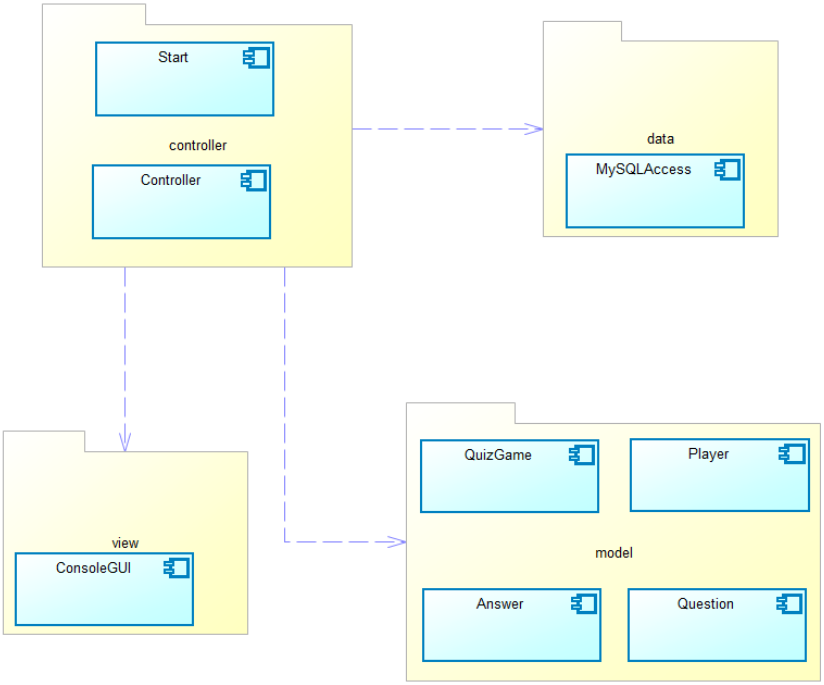


Figure 2 : diagramme des composants de l'application

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) throws FileNotFoundException,
ClassNotFoundException, ParseException, IOException, SQLException {
    //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(){

}
```

2 Implementation des classes metiers

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

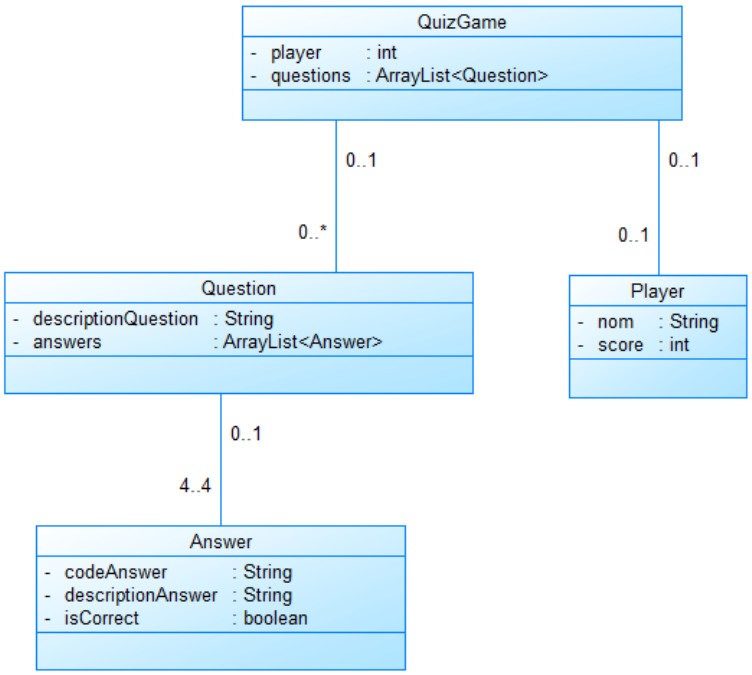


Figure 3 : diagramme des classes métiers

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

```
public class QuizGame {

   private Player myPlayer;

    private ArrayList<Question> questions;
    private ArrayList<Integer> listeIdQuestions;

    private Controller monController;

    public QuizGame(Controller aController, Player player, ArrayList<Question>
questions) {
         this.monController = aController;
        this.myPlayer = player;
        this.questions = questions;
    }

   public Player getMyPlayer() {
    return myPlayer;
   }

   public void setMyPlayer(Player myPlayer) {
    this.myPlayer = myPlayer;
   }

   public ArrayList<Question> getQuestions() {
        return questions;
    }

    public void setQuestions(ArrayList<Question> questions) {
        this.questions = questions;
    }
```

```
    public Boolean isCorrectThisAnswer(Question question, int idAnswer) {
        if (question.getAnswers().get(idAnswer).getIsCorrect()) {
            this.myPlayer.setMyScore(this.myPlayer.getMyScore() + 10);
        return true;
        } else {
            return false;
        }
    }
}
```

3 ACCES AUX DONNEES

3.1 LIRE UN FICHIER PROPERTIES[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")));
```

3.2 LIRE ET ECRIRE DANS UNE BASE DE DONNEES – JDBC[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.

[3] https://fr.wikipedia.org/wiki/.properties
https://www.techiedelight.com/fr/read-properties-files-java/

[4] https://devstory.net/10167/java-jdbc#a12649

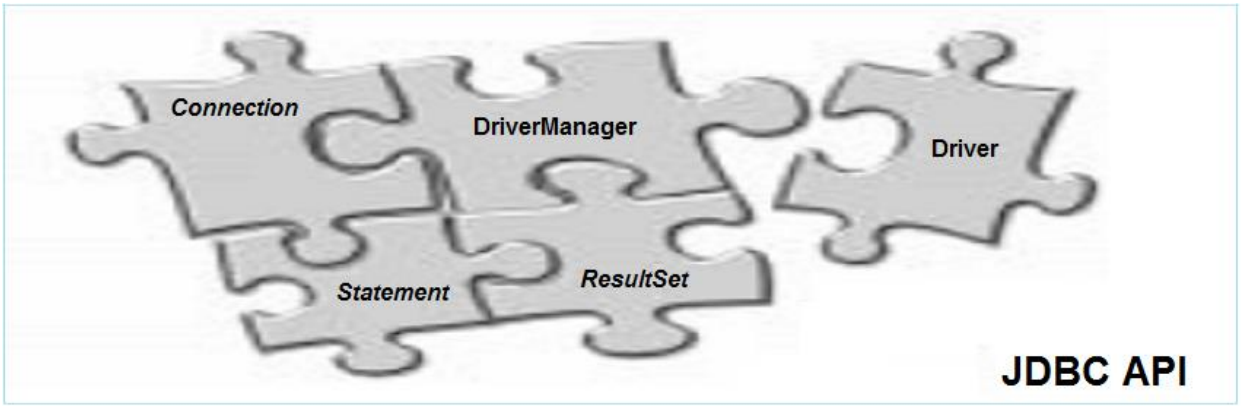


Figure 4 : principe de fonctionnement de l'API JDBC

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.

3.2.1 SE CONNECTER A UNE BASE DE DONNEES

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

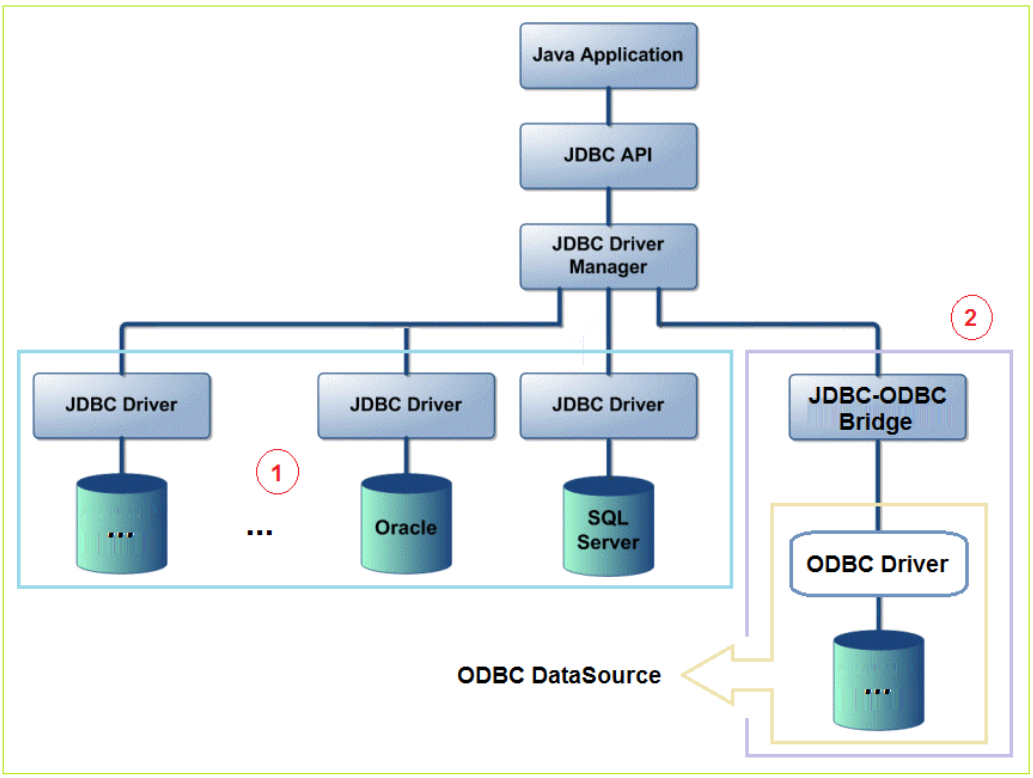


Figure 5 : les différentes méthodes de connexion avec JDBC

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();
```

3.2.2 Envoyer une requete 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);
```

3.2.3 Manipuler le resultat

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));
}
```

4 GERER UNE COLLECTION D'OBJET[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 "Answer" instanciés à partir des valeurs lues dans la base de données afin de stocker toutes les réponses d'une question.

```
[…]
   private ArrayList<Answer> getAnswersFromQuestion(int id)
         throws   FileNotFoundException,   ClassNotFoundException,   IOException,
SQLException {

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

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

    try (Statement st = conn.createStatement()) {
         ResultSet resultSet = st.executeQuery(strSqlQuestion);
         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));
         }
         return answers;
    }

   }
[…]
```

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.

[5] https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-collections-d-objets
http://www.jmdoudoux.fr/java/dej/chap-collections.html

Notez bien : la comparaison entre deux chaînes de caractère à ne pas confondre avec la comparaison entre deux pointeurs sur deux chaines.[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.

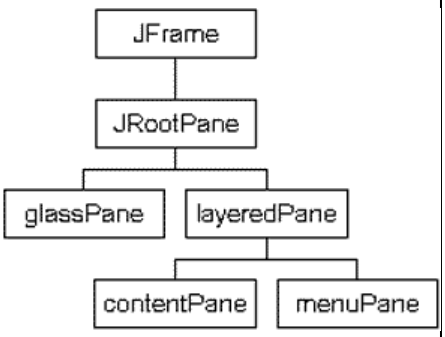
```
int comparaison = "Hello".compareTo("World");
System.out.println(comparaison); /* Nombre négatif car H < W */
```

5 L'Interface Homme Machine

5.1 Gerer plusieurs JPanel dans une JFrame[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.



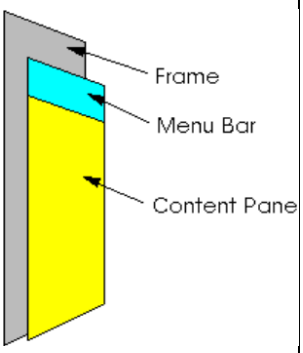
Les composants top-level possèdent un content pane qui contient tous les composants visibles d'un top-level. Un composant top-level peut aussi contenir une barre de menu

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 :

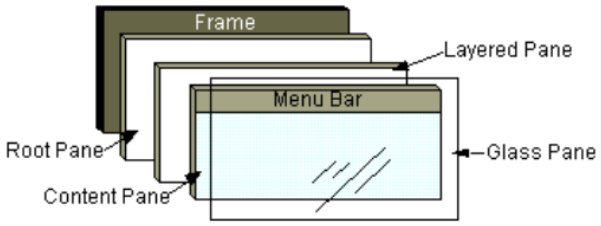
---

[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

[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

- 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…

5.1.1 Layout[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.

5.2 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. En effet, les JPanel se succèdent en devenant visibles puis invisibles. Prenons comme exemple le JPanel permettant de créer le quiz, où il est demandé le nom du joueur et le nombre de question. Une fois le bouton "Commencer"

[8] http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html

appuyé, le JPanel va devenir invisible, et le JPanel affichant la question avec les réponses va devenir visible.

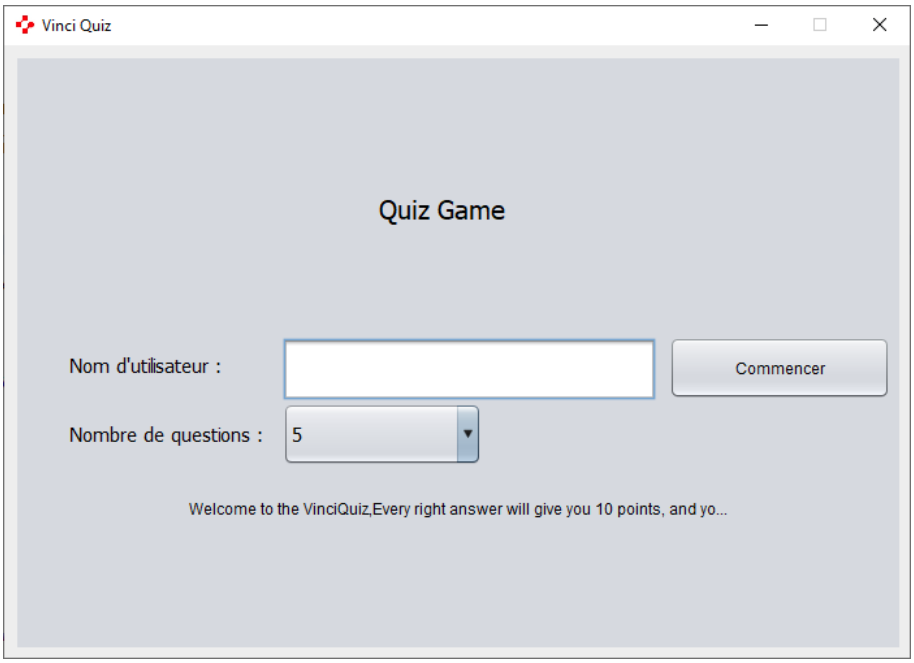


Figure 6 : JPanel permettant de créer le quiz

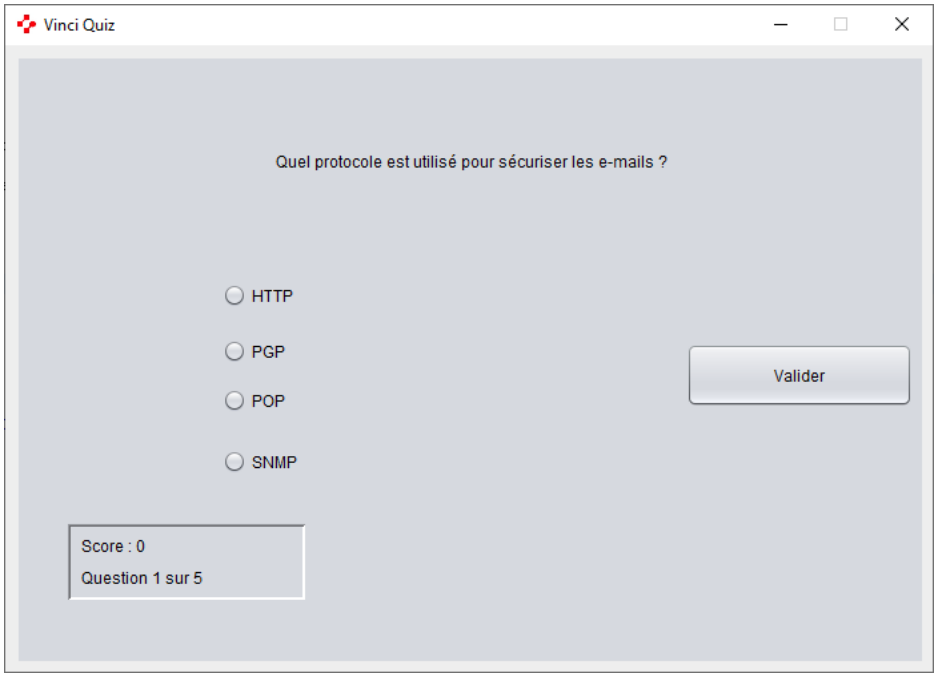


Figure 7 : JPanel montrant la question avec les réponses

Ce changement s'effectue à l'aide du code suivant :

```
    private void setVisible() {
     System.out.println("Start game : \nNom : " + textNom.getText() + "\nNombre de
  question : "
                 + listeNbQuestionSolo.getSelectedItem());
     lblErrorStartQuiz.setText("");
     pnlSoloCreateGame.setVisible(false);
     pnlDisplayQuiz.setVisible(true);
     }
```

5.2.1 LES BOUTONS POUR VALIDER LES SAISIES[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].
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].

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 !");
```

[9] http://java.developpez.com/faq/gui?page=Les-listeners
http://codes-sources.commentcamarche.net/faq/369-swing-partie-2-actionlistener-listener-jbutton

[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

[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

```
        }
    }

[…] }
```

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());
                        }
                });
        }
}
```

5.2.2 LES BOUTONS RADIO POUR LA SELECTION DE LA REPONSE[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);
```

[12] https://docs.oracle.com/javase/tutorial/uiswing/components/button.html#radiobutton

```
        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");
     }
    }
```

5.2.3 LES LISTES DEROULANTES[13]

En Swing, il existe 2 sortes de listes déroulantes :

[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

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 :

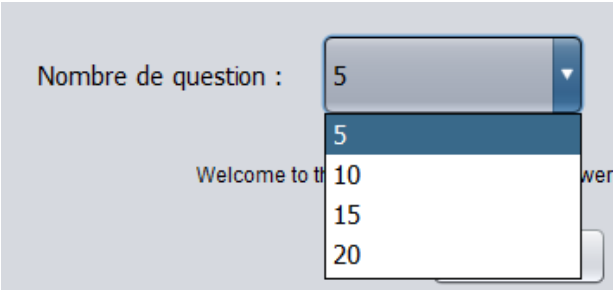


Figure 8 : liste déroulante déterminant le nombre de question du quiz

6 JAR - ARCHIVAGE - DEPLOIEMENT ET SIGNATURE[14]

14 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/

6.1 Le package JAR archive

Il s'agit d'une archive contenant des fichiers Java compilés (des fichiers .class). Le terme de Jar singifie 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.

6.1.1 Generation d'un jar file assistee 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.

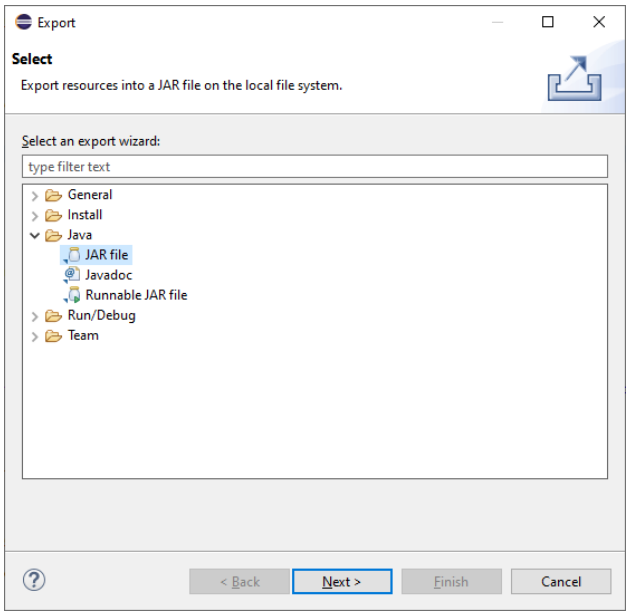


Figure 9 : génération d'un jar file à l'aide d'éclipse - étape 1

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 :

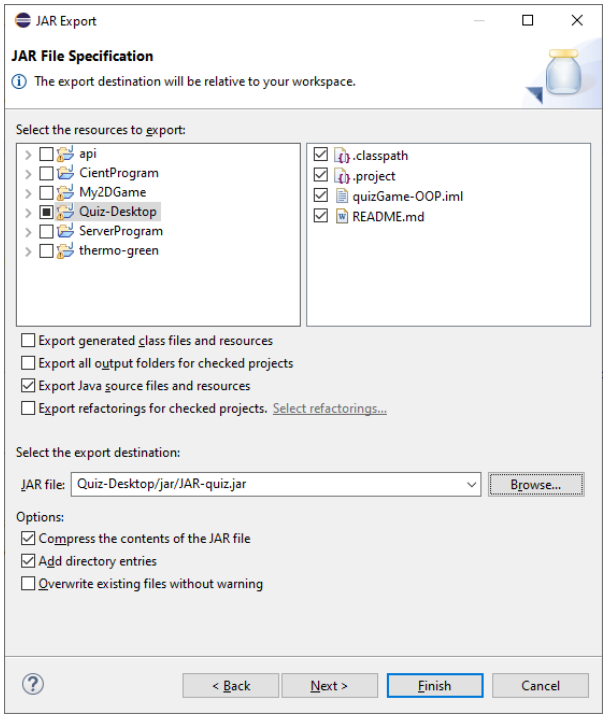


Figure 10 : génération d'un jar file à l'aide d'éclipse - étape 2

6.2 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.

6.2.1 Generation d'un JAR file assistee 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.

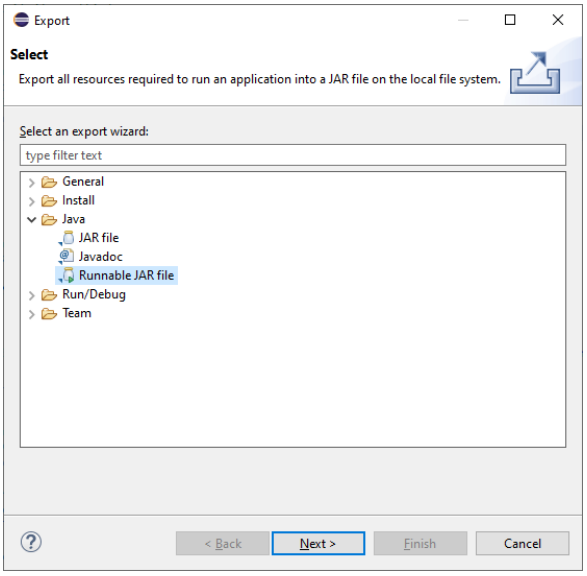


Figure 11 : génération d'un jar runnable à l'aide d'éclipse - étape 1

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 :

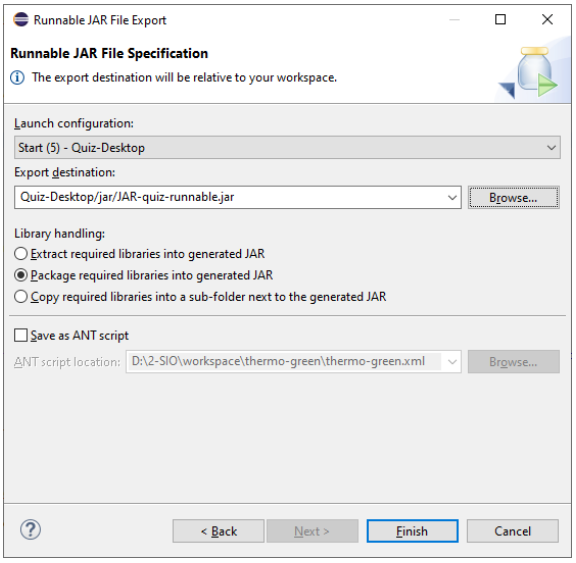


Figure 12 : génération d'un jar runnable à l'aide d'éclipse - étape 2

7 TABLE DES ILLUSTRATIONS