|  |
| --- |
| Documentation Développeur  De l’application JAVA pour la gestion des classes |

|  |
| --- |
| Par Amine NAKHIL, Loïc GUO et Quentin LIGNANI |



**Sommaire :**

**1. Page Admin\_Classe.java**

**2.Page Admin\_Menu.java**

**3. Page Admin\_viesco.java**

**4. Page Connexion.java**

**5. Page Eleves.java**

**6. Page Liste\_Eleve.java**

**7. Page lycee\_java.sql**

**8.Database.java,**

**9.Controller\_connexion.java,**

**10.Global.java**

1. Page Admin\_Classe.java

**Cette page est celle de la vue de l’administrateur sur les élèves. Elle apparait quand on clique dessus dans le menu initial de l’administrateur.**



**Nous importons les librairies de java nécessaires au code. En l’occurrence les librairies qui permettent de faire la jonction avec la base de données. Ensuite, nous importons les packages de l’IDE Eclipse nécessaires comme pour les listes déroulantes, les widgets, les boutons et l’implantation de texte.**



**Déclaration de la méthode « Admin\_Classe » qui hérite de la méthode « Global » dans le cadre du langage JAVA. On déclare ensuite les attributs shlClasse, Database, Connexion et Class. L’attribut shlClasse est accessible en protégé tandis que Classe est en privé. ……………………………………………………………………………………………**



**On déclare une classe de type void accessible en public. …………………………………………**



**On déclare une méthode void en accès protégé. Puis dans un premier temps, on place le texte « Classe », puis un formulaire sous forme de label « Classe : ». Ensuite, on affiche un menu déroulant qui va interroger la base de données pour obtenir la liste des classes au sein du menu déroulant.**



**On fait une boucle dans l’objectif de capturer ce que l’utilisateur a choisi dans le menu déroulant (une absence, une sanction ou un retard). Il faut que l’utilisateur valide son choix.**



**On ajoute un formulaire sous forme de label, dans lequel on écrit « libellé » tout en encodant de la manière nécessaire.**



**On ajoute un bouton de suppression dans lequel, il est logiquement écrit « Supprimer ». On ajoute également un bouton da validation avec écrit « Valider ». On ajoute un formulaire sous forme de label avec écrit dessus « Libellé : » en appliquant l’encodage. On instaure ensuite évidement des boutons qui permettent à l’utilisateur de nous faire part de son choix. On ajoute ensuite un bouton pour ajouter une classe avec écrit logiquement « Ajouter une classe ».**

2.Page Admin\_Menu.java



**Nous importons les librairies de java nécessaires au code. En l’occurrence les librairies de l’IDE Eclipse nécessaires. Par la suite, nous importons les librairies nécessaires au concepteur d’Eclipse Windows Builder. Enfin, nous importons les packages nécessaires comme pour les labels, les widgets, les boutons et l’implantation de texte.**



**On déclare une classe qui porte le nom du fichier à savoir « Admin\_Menu ». Elle hérité de « Global ». Elle est en accès public. On déclare ensuite l’attribut « Menu Admin » de type Shell en accès protégé.**



**Déclaration d’une nouvelle méthode en accès public. ………………..**



**Déclaration d’une nouvelle méthode en accès protégé. On déclare un nouvel attribut de type objet de nom MenuAdmin. Cela permet la déclaration d’un bouton, on y affiche le texte « Menu Admin ». On fait la même chose avec le nom de l’objet « Menu » et en y affichant le texte « Menu ».**



**On met un bouton qui affiche « Gestion utilisateurs ». Ce bouton doit rediriger vers la page « Liste\_Eleve » dans une nouvelle fenêtre. On met tout cela dans une procédure, et on y met la gestion d’erreur pour que si cela échoue, il y ait un message d’erreur.**



**On met un bouton qui affiche « Gestion Eleve ». Ce bouton doit rediriger vers la page « Liste\_Eleve » dans une nouvelle fenêtre. On met tout cela dans une procédure, et on y met la gestion d’erreur pour que si cela échoue, il y ait un message d’erreur.**



**On insère un bouton avec écrit « Modifier le planning »**

1. Page Admin\_viesco.java



**Nous importons les librairies de java nécessaires au code. En l’occurrence les librairies qui permettent de faire la jonction avec la base de données qui permettent l’interaction et l’interrogation des données contenues dans la base. Ensuite, nous importons les packages de l’IDE Eclipse nécessaires comme pour les listes déroulantes, les widgets, les boutons et l’implantation de texte, l’implantation de tableaux, de colonnes et d’éléments de tableaux.**



**On déclare la classe « Admin\_viesco » qui porte également le nom du fichier. Cette méthode est celle du fichier entier. Elle hérite de la méthode « Global ». On déclare également l’attribut ListeDesElves de type Shell et en accès protégé. On déclare également l’attribut « Table » de type tableau et en accès uniquement dans la méthode.**



**On déclare une méthode en accès public. ……………………………………………….**



**On déclare une méthode à l’intérieur du fichier, c’est une méthode void en accès protégé. On instaure un bouton sur lequel est écrit « Liste des Eleves ».**



**On met un widget composite et on indique son emplacement.**



**On déclare un formulaire de type Label, on indique les caractéristiques du texte comme la police d’écriture ainsi que sa couleur. On indique également l’emplacement du label. Le Label se nomme « Nom\_Eleve » et il affiche « Nom » à l’utilisateur.**



**On déclare un formulaire de type Label, on indique les caractéristiques du texte comme la police d’écriture ainsi que sa couleur. On indique également l’emplacement du label. Le Label se nomme « Prenom\_Eleve » et il affiche « Prenom » à l’utilisateur.**





**On déclare un tableau avec SWT. On y insère une colonne « Nom » et un élément « test ».**



**On ajoute un bouton pour modifier le profil, naturellement, on affiche « Modifier votre profil » en son sein.**



**On se connecte à notre base de données. On envoie une requête SQL pour récupérer l’ensemble du contenu des colonnes dans la table Eleve. En clair, on demande l’accès à une liste d’élèves. On demande spécifiquement les colonnes « nom » et « prenom », car ce sont les informations recherchées. On met une gestion d’erreur en cas d’erreur de connexion à la base de données.**



**On demande, si l’utilisateur clique sur le bouton pour ajouter une sanction ou une absence, de le rediriger vers la page « form ». On n’oublie pas la gestion d’erreur.**

1. Page Connexion.java



**Nous importons les librairies de java nécessaires au code. En l’occurrence les librairies qui permettent la gestion des exception SQL. Ensuite, nous importons les packages de l’IDE Eclipse nécessaires pour les widgets tel que les boutons et l’implantation de texte. Mais aussi avec les librairies de l’IDE Eclipse nécessaires pour la gestion des évènements**



**Cette méthode s’exécute comme partie du bloc d’instruction de la page. Cette méthode sert à y incorporer la gestion d’erreur pour la redirection vers la page de connexion. Dans le Try, on insère le code qui permet de rediriger vers la page de connexion et puis si cela échoue, dans le catch, on insère le code qui permet le message d’erreur.**



**Ouverture de l’interface de connexion.**



**On déclare une méthode void en accès protégé. On nomme l’en tête de la fenêtre ouverte.**



**En vert et en haut de la fenêtre, on affiche comme titre « Ma Gestion ». On affiche à l’utilisateur « Identifiant » et en dessous « Mot de passe » sous forme de texte. On prépare le Gestion d’erreur en déclarant l’affichage du message d’erreur en cas de mauvaise saisie. Ce message n’est pas visible par défaut.**



**Le bouton de validation contrôle la validité des informations de connexions fournies par l’utilisateurs, son identifiant et son mot de passe. On y met aussi une gestion d’erreur.**

1. Page Eleves.java



**On déclare la classe « Eleves » qui hérite de « Global ». On déclare ensuite les attributs Shell (de type shell et en accès protégé), BtsSlam (de type bouton et en accès privé), TerminaleStid (de type bouton et en accès privé). Puis mardi, mercredi, lundi, jeudi et vendredi tous de type tableau et en accès privé.**



**On déclare la méthode « Eleve ». En son sein, on déclare l’attribut shell de type shell et en accès protégé et les attributs table, table\_1, table\_2 et table\_3. Tous de type tableau et en accès privé.**



**Ouverture de l’interface de connexion.**



**On déclare une méthode void en accès protégé. Ensuite formalise notre intention d’interroger la base de données. Puis on renseigne notre requête avec le langage SQL. On demande d’accéder à la table utilisateur de la base de données, plus précisément, on demande l’accès aux colonnes nom, au prénom et aux identifiants parmi lesquels les noms correspondent aux noms liés aux id que l’on possède.**



**On ajoute un label avec affiché « Classe : » de sorte à informer l’utilisateur de nos intentions. On fait la même chose pour afficher « Bonjour » au centre en haut, puis le nom de l’utilisateur. On fait la même chose avec « Cours : » pour afficher les cours.**



**On institue trois labels, un « Informatique », l’autre « Horaires : » et le dernier « 13h -15h ». On lie le label horaire au planning. Ensuite, nous ajoutons trois boutons, l’un pour ajouter un retard, un autre pour ajouter une absence et un dernier pour ajouter une sanction.**



**On institue un nouveau tableau avec Windows Builder. On lui attribue une colonne « Eleve » par le même biais. On déclare ensuite trois éléments du tableau qui serons dans la table « Eleve ». Ensuite, on déclare un nouveau tableau qui se nomme « table\_1 ».**



**On ajoute une colonne de « table\_2 » qui se nomme « Absence ». On institue ensuite un tableau que l’on nomme « table\_3 » avec windows Builder. Enfin, on lui ajoute une colonne appelée « Sanction »**

1. Page Liste\_Eleve.java



**On institue une nouvelle classe qui portera naturellement le nom de « Liste\_Eleve », elle hérite de « Global ». On déclare un attribut de type Shell qui se nomme « ListeEleve » en accès protégé. On déclare un autre attribut : « Table », de type tableau et en accès privé. On déclare un autre attribut : « Database », et de type objet.**



**On déclare une méthode vois en accès protégé. On déclare l’attribut « ListeEleve » de type objet. Avec windows Builder. On impute le texte « Liste des Elèves » avec le bon encodage.**



**On met un composite (avec windows Builder) en face. Pour que l’utilisateur puisse interagir avec la machine.**



**On met un label composite pour sélectionner un** **élève. En indiquant ses caractéristiques.**



**Sous forme de liste, on ajoute un label, on indique ses caractéristiques à travers le package Windows Builder et on l’indique à l’utilisateur en affichant « Liste des Elèves ». Cela vise à afficher les élèves.**



**Sous forme de liste, on ajoute un label, on indique ses caractéristiques à travers le package Windows Builder et on l’indique à l’utilisateur en affichant « Classes : ». Cela vise à afficher les classes respectives des élèves.**



**On déclare trois colonnes du tableau libelle. Une pour le nom, une pour le prénom et une autre pour l’identifiant. Les colonnes sont nommées en haut et y figures une chaine de caractères indiquant à l’utilisateur pour l’identifiant. Les colonnes sont nommées en haut et y figures une chaine de caractères indiquant à l’utilisateur le contenu de la colonne. On déclare un bouton, sans encore décrire ce qu’il implique en arrière-plan. Sur ce bouton, il est marqué, la chaine de caractère « Retour ». En fait, il sers à fermer la fenêtre et à revenir à celle précédente.**

6. Page lycee\_java.sql

**Cette page est un fichier en SQL, il contient la base de données du projet, elle se nomme « lycee\_java ».**



**Cette commande sert à fonder dans la base une table nommée « classe » dans le cas où elle n’existerait pas. On déclare ensuite quatre colonnes au sein de la base de données. Une colonne « id » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur), une colonne « libelle » (de type varchar à trente caractères et qui ne peut pas être nul), une colonne « id\_prof\_principal » (de type entier à onze caractères et qui est vide sans instruction inverse), et une colonne « undelatable » (de type entier de 0 à 255 à un caractère et avec par défaut le caractère « 0 »). On ajoute la colonne « id » en clef primaire pour l’identifier de manière unique à travers la table et permettre l’incrémentation automatique.**



**Dans la table « classe », on ajoute pour toutes les colonnes préalablement déclarées. On indique que le libellé « Non attribué » pour la première ligne, « BTS SIO SLAM » pour la seconde et « test » pour la troisième, puis une chaine de caractère vide pour les deux dernières. On met la colonne « id\_prof\_principal » en NULL pour toute les lignes, et la colonne « id » s’attribue automatiquement en vertu de l’instruction AUTO INCREMENT. Il ne reste plus que la colonne « undeletable » qui est partout le caractère par défaut, sauf la première ligne ou c’est « 1 ».**



**Cette commande sert à fonder dans la base une table nommée « eleve » dans le cas où elle n’existerait pas. On déclare ensuite quatre colonnes au sein de la base de données. Une colonne « id » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère), une colonne « nom » (de type varchar à trente caractères maximums et qui ne peut pas être nul), une colonne « prenom » (de type varchar à trente caractères maximums et qui ne peut pas être nul), et une colonne « id\_classe » (de type entier à 11 caractères et qui ne peut pas être nul).**



**Dans la table « eleve », on ajoute quatre élèves, LIGNANI quentin, GUO Loïc, NAKHIL Amine et Martini Americano. Leurs colonnes « id » sont auto-incrémentées, ils sont tous dans la classe identifiée par l’entier « 2 », sauf Americano qui est dans celle identifiée par le « 1 ».**



**Dans la table « heure, on insère les lignes suivantes pour la colonne « libelle » : « 8 h », « 9h », « 10h », « 11h », « 12h », « 14h », « 15h », « 16h » et « 17h ». La colonne « id » s’auto incrémente en clef étrangère.**



**On déclare une table « jour » dans le cas ou elle n’existerait pas. On y insère deux colonnes, l’une « id » de type entier à onze caractères sans possibilité de virginité et la valeur est auto-incrémentée dans une clef étrangère et une autre colonne nommée « libelle », de type varchar à 20 charactères maximum sans possibilité de virginité.**



**On insère dans la table « jour » tous les jours ouvrés de la semaine, à raison d’un jour par ligne. Ces jours sont stockés dans la colonne « libelle » et la colonne « id » est auto-incrémentée.**



**Cette commande sert à fonder dans la base une table nommée « planning » dans le cas où elle n’existerait pas. On déclare ensuite cinq colonnes au sein de la base de données. Une colonne « id » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère), une colonne « id\_jour » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère), une colonne « id\_heure » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère), une colonne « id\_classe » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère), une colonne « id\_professeur » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère),**



**On rempli trois ligne de la table « planning ». Mais toutes les colonnes sont en auto-incrémente, on laisse faire.**



**On fonder dans la base une table nommée « utilisateur » dans le cas où elle n’existerait pas. On déclare ensuite quatre colonnes au sein de la base de données. Une colonne « id » (de type entier à 11 caractères, qui ne peut pas être vide), une colonne « prenom » (de type varchar à 30 caractères maximum, qui ne peut pas être vide), une colonne « email » (de type varchar à 40 caractères maximum, qui ne peut pas être vide), une colonne « mdp » (de type varchar à 100 caractères maximum, qui ne peut pas être vide), une colonne « matiere » (de type varchar à 30 caractères maximum, qui est vide par défaut) et une colonne « role » (de type varchar à 10 caractères maximum, qui ne peut pas être vide et qui par défaut à la valeur de la chaine « professeur »). On marque le lien avec la clef étrangère pour retrouver la colonne « identifiant » déclarée précédemment.**



**On fonder dans la base une table nommée « vie scolaire » dans le cas où elle n’existerait pas. On déclare ensuite quatre colonnes au sein de la base de données. Une colonne « id » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère), une colonne « id\_eleve » (de type entier à 11 caractères, qui ne peut pas être vide et qui s’auto assigne la valeur en clef étrangère), une colonne « type » (de type varchar à 30 caractères maximum, qui ne peut pas être vide), une colonne « date » (de type date et qui ne peut pas être vide) et une colonne « justification » (de type varchar à 30 caractères maximum, qui ne peut pas être vide). On marque le lien avec la clef étrangère pour retrouver la colonne « id\_eleve » déclarée précédemment.**



**On ajoute une sanction ou une justification d’absence/retard dans la table « vie\_scolaire ». On renseigne l’identifiant de l’élève, en l’occurrence 4, afin que la clef étrangère détermine l’élève, la date, la nature de l’enregistrement en l’occurrence une absence et la justification en l’occurrence la Maladie.**



**On modifie la table « classe ». On y ajoute une contrainte de type clef étrangère. Avec la colonne « id\_prof\_principal ». On la lie à la colonne « fk\_prof\_prin ».**



**On modifie la table « eleve ». On y ajoute une contrainte de type clef étrangère. On y ajoute une contrainte de type clef étrangère. Avec la colonne « id\_prof\_principal ». On la lie à la colonne « fk\_classe ».**



**On modifie la table « planning ». On y ajoute quatre contraintes de type clef étrangère. On y ajoute une contrainte de type clef étrangère. Avec la colonne « id\_classe », « id\_heure », « id\_jour » et « id\_professeur ». On les lies respecivement aux colonnes « fk\_classe2 », « fk\_heure », « fk\_jour » et « fk\_prof ».**



**On modifie la table « vie\_scolaire ». On y ajoute une contrainte de type clef étrangère. On y ajoute une contrainte de type clef étrangère. Avec la colonne « id\_eleve ». On la lie à la colonne « fk\_eleve ».**

8.Database.java

**Cette page a principalement pour but d’énoncer les principes qui encadreront l’interaction du programme avec la base de données.**



**On importe les librairies de JAVA nécessaires. Pour cette page ce sont bien sur les librairies qui font le pont avec le langage SQL en vue d’interagir avec la base de données.**



**On déclare la classe principale en accès qui public qui a le nom « Database ». Puis, on déclare une méthode typée « Connection » : « DbConnection » et en accès public. Ensuite, on renseigne les premières formalités nécessaires à la connexion à la bdd. En l’occurrence, son chemin d’accès et les identifiants avec lesquels on utilise SQL.**



**Une fois que les informations préliminaires ont étés mises en bonne et due forme, on exécute la requête en l’envoyant.**



**Toutefois, on met une gestion d’erreur, dans le cas ou la requête précédente n’est pas concluante, et on informe l’utilisateur que la bdd n’a pas pu être contactée. Dans tous les cas, à la fin de la méthode, on ne retourne rien.**



**On déclare une nouvelle méthode, de type ResultSet et nommée Request. Elle intervient directement en cas d’application réussie de la méthode « Connexion ». On demande ensuite à récupérer les données conformément à ce que l’on a ordonné au cas par cas, selon la requête et en fonction de la page. Si cela réussi, on retourne les informations issues de cette requête. Dans le cas ou cela ne fonctionne pas, on ne fait rien comme exception. Mais dans tous les cas, on retournera une valeur nulle.**

9.Controller\_connexion.java



**On déclare la classe de la page, elle est nommée « Controller\_connexion », elle hérite de la classe « Global ». Ensuite, on déclare une nouvelle méthode, de type booléen et nommée « Connexion ». Cette méthode prend le soin de se procurer les identifiants de connexion de l’utilisateur au préalable. Ensuite, le programme prend contact avec la bdd à travers une méthode extérieure qui est appelée pour l’occasion. Ensuite, on envoie une requête à la base de données pour se renseigner sur l’existence de données de connexions d’un utilisateurs semblables à celles entrées par l’utilisateur. On déclare une variable « role », de type chaine de caractère. Et enfin, on appelle la méthode chargée d’interagir avec la base de données.**

****

**On demande à ce que la requête soit répétée indéfiniment jusqu’à ce qu’il y ait une occurrence pour un utilisateur étant administrateur. Toujours avec les mêmes identifiants.**

****

**En cas d’occurrence, on modifie la variable globale relative à l’administrateur de sorte à ce qu’y soit inscrit un booléen vrai.**



**La personne étant habilitée, on lui fait se connecter à l’interface spécifique aux administrateurs. On ferme donc la fenêtre en cours. On ouvre la page « window\_admin » et on retourne le booléen faux pour cette condition de la méthode. En cas d’erreur, nous avons un message à cet effet.**



**Dans le cas inverse, les identifiants existent mais la personne n’est pas habilitée à être administrateur. Dans ce cas, on ferme la fenêtre en cours pour ouvrir la page « Planning\_prof ». Et on retourne la valeur booléenne fausse. Là aussi nous disposons d’une gestion d’erreur. Dans tous les càs, nous retournons la valeur booléenne vraie.**

10.Global.java



**Cette page est celle des variables globales. On importe le package nécessaire. Puis on déclare une classe « Global » en accès public. On déclare ensuite les variables pour les identifiants, l’administrateur, le nom, les élèves et la sélection. Elles sont toutes des chaines de caractères, sauf celle relative à l’administrateur qui es booléenne, initialisé à la valeur fausse. Pour les variables relatives aux élèves et aux identifiants, on les initialise avec une chaine de caractère vide et pour toute les autres, on ne leur assigne pas de valeurs.**