OperationeditorpaneVirement.fxml : (Créer un fichier fxml)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.ComboBox?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.TextField?>

<?import javafx.scene.layout.BorderPane?>

<?import javafx.scene.layout.ColumnConstraints?>

<?import javafx.scene.layout.GridPane?>

<?import javafx.scene.layout.HBox?>

<?import javafx.scene.layout.RowConstraints?>

<BorderPane maxHeight=*"-Infinity"* maxWidth=*"-Infinity"* minHeight=*"-Infinity"* minWidth=*"-Infinity"* prefHeight=*"250.0"* prefWidth=*"500.0"* xmlns=*"http://javafx.com/javafx/8.0.171"* xmlns:fx=*"http://javafx.com/fxml/1"* fx:controller=*"application.view.OperationEditorPaneController"*>

<top>

<Label fx:id=*"lblMessage"* alignment=*"CENTER"* maxWidth=*"1.7976931348623157E308"* styleClass=*"borderlight"* text=*"Informations "* BorderPane.alignment=*"CENTER"*>

<BorderPane.margin>

<Insets bottom=*"10.0"* left=*"10.0"* right=*"10.0"* top=*"10.0"* />

</BorderPane.margin>

<padding>

<Insets bottom=*"10.0"* left=*"10.0"* right=*"10.0"* top=*"10.0"* />

</padding>

</Label>

</top>

<center>

<GridPane styleClass=*"borderlight"* BorderPane.alignment=*"CENTER"*>

<columnConstraints>

<ColumnConstraints hgrow=*"SOMETIMES"* maxWidth=*"190.0"* minWidth=*"10.0"* percentWidth=*"0.0"* prefWidth=*"190.0"* />

<ColumnConstraints hgrow=*"SOMETIMES"* maxWidth=*"303.0"* minWidth=*"10.0"* prefWidth=*"278.0"* />

</columnConstraints>

<rowConstraints>

<RowConstraints minHeight=*"10.0"* prefHeight=*"30.0"* vgrow=*"SOMETIMES"* />

<RowConstraints minHeight=*"10.0"* prefHeight=*"30.0"* vgrow=*"SOMETIMES"* />

<RowConstraints minHeight=*"10.0"* prefHeight=*"30.0"* vgrow=*"SOMETIMES"* />

</rowConstraints>

<BorderPane.margin>

<Insets bottom=*"10.0"* left=*"10.0"* right=*"10.0"* top=*"10.0"* />

</BorderPane.margin>

<padding>

<Insets bottom=*"10.0"* left=*"10.0"* right=*"10.0"* top=*"10.0"* />

</padding>

<children>

<Label alignment=*"CENTER\_RIGHT"* maxWidth=*"1.7976931348623157E308"* text=*"Type d'opération"*>

<GridPane.margin>

<Insets right=*"20.0"* />

</GridPane.margin>

</Label>

<Label fx:id=*"lblMontant"* alignment=*"CENTER\_RIGHT"* layoutX=*"21.0"* layoutY=*"41.0"* maxWidth=*"1.7976931348623157E308"* text=*"Montant"* GridPane.rowIndex=*"1"*>

<GridPane.margin>

<Insets right=*"20.0"* />

</GridPane.margin>

</Label>

<ComboBox fx:id=*"cbTypeOpe"* maxWidth=*"1.7976931348623157E308"* GridPane.columnIndex=*"1"* />

<TextField fx:id=*"txtMontant"* layoutX=*"171.0"* layoutY=*"23.0"* GridPane.columnIndex=*"1"* GridPane.rowIndex=*"1"* />

<Label fx:id=*"lblMontant1"* alignment=*"CENTER\_RIGHT"* maxWidth=*"1.7976931348623157E308"* text=*"ID du compte à virer"* GridPane.rowIndex=*"2"*>

<GridPane.margin>

<Insets right=*"20.0"* />

</GridPane.margin>

</Label>

<TextField fx:id=*"txtCompteV"* onKeyPressed=*"#getCompteV"* GridPane.columnIndex=*"1"* GridPane.rowIndex=*"2"* />

</children>

</GridPane>

</center>

<bottom>

<HBox alignment=*"CENTER\_RIGHT"* BorderPane.alignment=*"CENTER\_RIGHT"*>

<children>

<Button fx:id=*"btnOk"* mnemonicParsing=*"false"* onAction=*"#doAjouter"* text=*"Valider"*>

<HBox.margin>

<Insets right=*"20.0"* />

</HBox.margin>

</Button>

<Button fx:id=*"btnCancel"* mnemonicParsing=*"false"* onAction=*"#doCancel"* text=*"Annuler"* />

</children>

<BorderPane.margin>

<Insets bottom=*"10.0"* left=*"10.0"* right=*"10.0"* top=*"10.0"* />

</BorderPane.margin>

</HBox>

</bottom>

</BorderPane>

OperationeditorpaneController.java : remplacer tout le fichier

package application.view;

import java.net.URL;

import java.util.Locale;

import java.util.ResourceBundle;

import application.DailyBankState;

import application.tools.AlertUtilities;

import application.tools.CategorieOperation;

import application.tools.ConstantesIHM;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.control.Button;

import javafx.scene.control.ComboBox;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import javafx.stage.WindowEvent;

import model.data.CompteCourant;

import model.data.Operation;

public class OperationEditorPaneController implements Initializable {

// Etat application

private DailyBankState dbs;

// Fenêtre physique

private Stage primaryStage;

// Données de la fenêtre

private CategorieOperation categorieOperation;

private CompteCourant compteEdite;

private Operation operationResultat;

// Manipulation de la fenêtre

public void initContext(Stage \_primaryStage, DailyBankState \_dbstate) {

this.primaryStage = \_primaryStage;

this.dbs = \_dbstate;

this.configure();

}

private void configure() {

this.primaryStage.setOnCloseRequest(e -> this.closeWindow(e));

}

public Operation displayDialog(CompteCourant cpte, CategorieOperation mode) {

this.categorieOperation = mode;

this.compteEdite = cpte;

switch (mode) {

case DEBIT:

String info = "Cpt. : " + this.compteEdite.idNumCompte + " "

+ String.format(Locale.ENGLISH, "%12.02f", this.compteEdite.solde) + " / "

+ String.format(Locale.ENGLISH, "%8d", this.compteEdite.debitAutorise);

this.lblMessage.setText(info);

this.btnOk.setText("Effectuer Débit");

this.btnCancel.setText("Annuler débit");

ObservableList<String> list = FXCollections.observableArrayList();

for (String tyOp : ConstantesIHM.OPERATIONS\_DEBIT\_GUICHET) {

list.add(tyOp);

}

this.cbTypeOpe.setItems(list);

this.cbTypeOpe.getSelectionModel().select(0);

break;

case CREDIT:

String infoC = "Cpt. : " + this.compteEdite.idNumCompte + " "

+ String.format(Locale.ENGLISH, "%12.02f", this.compteEdite.solde);

this.lblMessage.setText(infoC);

this.btnOk.setText("Effectuer Crédit");

this.btnCancel.setText("Annuler crédit");

ObservableList<String> listC = FXCollections.observableArrayList();

for (String tyOpC : ConstantesIHM.OPERATIONS\_CREDIT\_GUICHET) {

listC.add(tyOpC);

}

this.cbTypeOpe.setItems(listC);

this.cbTypeOpe.getSelectionModel().select(0);

break;

case VIREMENT:

String infoV = "Cpt. : " + this.compteEdite.idNumCompte + " "

+ String.format(Locale.ENGLISH, "%12.02f", this.compteEdite.solde);

this.lblMessage.setText(infoV);

this.btnOk.setText("Effectuer virement");

this.btnCancel.setText("Annuler virement");

ObservableList<String> listV = FXCollections.observableArrayList();

for (String tyOpV : ConstantesIHM.OPERATIONS\_VIREMENT\_GUICHET) {

listV.add(tyOpV);

}

this.cbTypeOpe.setItems(listV);

this.cbTypeOpe.getSelectionModel().select(0);

break;

}

// Paramétrages spécifiques pour les chefs d'agences

if (ConstantesIHM.isAdmin(this.dbs.getEmpAct())) {

// rien pour l'instant

}

this.operationResultat = null;

this.cbTypeOpe.requestFocus();

this.primaryStage.showAndWait();

return this.operationResultat;

}

// Gestion du stage

private Object closeWindow(WindowEvent e) {

this.doCancel();

e.consume();

return null;

}

// Attributs de la scene + actions

@FXML

private Label lblMessage;

@FXML

private Label lblMontant;

@FXML

private ComboBox<String> cbTypeOpe;

@FXML

private TextField txtMontant;

@FXML

private TextField txtCompteV;

@FXML

private Button btnOk;

@FXML

private Button btnCancel;

@Override

public void initialize(URL location, ResourceBundle resources) {

}

@FXML

public int getCompteV() {

return Integer.parseInt(txtCompteV.getText());

}

@FXML

private void doCancel() {

this.operationResultat = null;

this.primaryStage.close();

}

@FXML

private void doAjouter() {

switch (this.categorieOperation) {

case DEBIT:

// règles de validation d'un débit :

// - le montant doit être un nombre valide

// - et si l'utilisateur n'est pas chef d'agence,

// - le débit ne doit pas amener le compte en dessous de son découvert autorisé

double montant;

this.txtMontant.getStyleClass().remove("borderred");

this.lblMontant.getStyleClass().remove("borderred");

this.lblMessage.getStyleClass().remove("borderred");

String info = "Cpt. : " + this.compteEdite.idNumCompte + " "

+ String.format(Locale.ENGLISH, "%12.02f", this.compteEdite.solde) + " / "

+ String.format(Locale.ENGLISH, "%8d", this.compteEdite.debitAutorise);

this.lblMessage.setText(info);

try {

montant = Double.parseDouble(this.txtMontant.getText().trim());

if (montant <= 0)

throw new NumberFormatException();

} catch (NumberFormatException nfe) {

this.txtMontant.getStyleClass().add("borderred");

this.lblMontant.getStyleClass().add("borderred");

this.txtMontant.requestFocus();

return;

}

if (this.compteEdite.solde - montant < this.compteEdite.debitAutorise) {

info = "Dépassement du découvert ! - Cpt. : " + this.compteEdite.idNumCompte + " "

+ String.format(Locale.ENGLISH, "%12.02f", this.compteEdite.solde) + " / "

+ String.format(Locale.ENGLISH, "%8d", this.compteEdite.debitAutorise);

this.lblMessage.setText(info);

this.txtMontant.getStyleClass().add("borderred");

this.lblMontant.getStyleClass().add("borderred");

this.lblMessage.getStyleClass().add("borderred");

this.txtMontant.requestFocus();

return;

}

String typeOp = this.cbTypeOpe.getValue();

this.operationResultat = new Operation(-1, montant, null, null, this.compteEdite.idNumCli, typeOp);

this.primaryStage.close();

break;

case CREDIT:

// règles de validation d'un débit :

// - le montant doit être un nombre valide

// - et si l'utilisateur n'est pas chef d'agence,

// - le débit ne doit pas amener le compte en dessous de son découvert autorisé

double montantC;

this.txtMontant.getStyleClass().remove("borderred");

this.lblMontant.getStyleClass().remove("borderred");

this.lblMessage.getStyleClass().remove("borderred");

String infoC = "Cpt. : " + this.compteEdite.idNumCompte + " "

+ String.format(Locale.ENGLISH, "%12.02f", this.compteEdite.solde);

this.lblMessage.setText(infoC);

try {

montantC = Double.parseDouble(this.txtMontant.getText().trim());

if (montantC <= 0)

throw new NumberFormatException();

} catch (NumberFormatException nfe) {

this.txtMontant.getStyleClass().add("borderred");

this.lblMontant.getStyleClass().add("borderred");

this.txtMontant.requestFocus();

return;

}

String typeOpC = this.cbTypeOpe.getValue();

// ce genre d'operation n'est pas encore géré

this.operationResultat = new Operation(-1, montantC, null, null, this.compteEdite.idNumCli, typeOpC);

this.primaryStage.close();

break;

case VIREMENT:

// règles de validation d'un débit :

// - le montant doit être un nombre valide

// - et si l'utilisateur n'est pas chef d'agence,

// - le débit ne doit pas amener le compte en dessous de son découvert autorisé

double montantV;

this.txtMontant.getStyleClass().remove("borderred");

this.lblMontant.getStyleClass().remove("borderred");

this.lblMessage.getStyleClass().remove("borderred");

String infoV = "Cpt. : " + this.compteEdite.idNumCompte + " "

+ String.format(Locale.ENGLISH, "%12.02f", this.compteEdite.solde);

this.lblMessage.setText(infoV);

try {

montantV = Double.parseDouble(this.txtMontant.getText().trim());

if (montantV <= 0)

throw new NumberFormatException();

} catch (NumberFormatException nfe) {

this.txtMontant.getStyleClass().add("borderred");

this.lblMontant.getStyleClass().add("borderred");

this.txtMontant.requestFocus();

return;

}

String typeOpV = this.cbTypeOpe.getValue();

// ce genre d'operation n'est pas encore géré

this.operationResultat = new Operation(-1, montantV, null, null, this.compteEdite.idNumCli, typeOpV);

this.primaryStage.close();

break;

}

}

}

OperationManagementcomntroller : ajouter juste cette fonction

@FXML

**private** **void** doVirement() {

Operation op = **this**.om.enregistrerVirement();

**if** (op != **null**) {

**this**.updateInfoCompteClient();

**this**.validateComponentState();

}

}

OperationEditorPaneVirement.java : creer classe

package application.control;

import application.DailyBankApp;

import application.DailyBankState;

import application.tools.CategorieOperation;

import application.tools.StageManagement;

import application.view.OperationEditorPaneController;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.scene.layout.BorderPane;

import javafx.stage.Modality;

import javafx.stage.Stage;

import model.data.CompteCourant;

import model.data.Operation;

public class OperationEditorPaneVirement {

private Stage primaryStage;

private OperationEditorPaneController oepc;

public int getCompteVirement() {

return oepc.getCompteV();

}

public OperationEditorPaneVirement(Stage \_parentStage, DailyBankState \_dbstate) {

try {

FXMLLoader loader = new FXMLLoader(

OperationEditorPaneController.class.getResource("operationeditorpaneVirement.fxml"));

BorderPane root = loader.load();

Scene scene = new Scene(root, 500 + 20, 250 + 10);

scene.getStylesheets().add(DailyBankApp.class.getResource("application.css").toExternalForm());

this.primaryStage = new Stage();

this.primaryStage.initModality(Modality.WINDOW\_MODAL);

this.primaryStage.initOwner(\_parentStage);

StageManagement.manageCenteringStage(\_parentStage, this.primaryStage);

this.primaryStage.setScene(scene);

this.primaryStage.setTitle("Enregistrement d'une opération");

this.primaryStage.setResizable(false);

this.oepc = loader.getController();

this.oepc.initContext(this.primaryStage, \_dbstate);

} catch (Exception e) {

e.printStackTrace();

}

}

public Operation doOperationEditorDialog(CompteCourant cpte, CategorieOperation cm) {

return this.oepc.displayDialog(cpte, cm);

}

}

Operation Management.java :

**public** Operation enregistrerVirement() {

OperationEditorPaneVirement oep = **new** OperationEditorPaneVirement(**this**.primaryStage, **this**.dbs);

Operation op = oep.doOperationEditorDialog(**this**.compteConcerne, CategorieOperation.***VIREMENT***);

**if** (op != **null**) {

**try** {

AccessOperation ao = **new** AccessOperation();

AccessCompteCourant acc = **new** AccessCompteCourant();

**int** compteVise = oep.getCompteVirement();

CompteCourant compte;

compte = acc.getCompteCourant(compteVise);

ao.insertVirement(**this**.compteConcerne.idNumCompte, op.montant, op.idTypeOp, compte.idNumCompte );

} **catch** (DatabaseConnexionException e) {

ExceptionDialog ed = **new** ExceptionDialog(**this**.primaryStage, **this**.dbs, e);

ed.doExceptionDialog();

**this**.primaryStage.close();

op = **null**;

} **catch** (ApplicationException ae) {

ExceptionDialog ed = **new** ExceptionDialog(**this**.primaryStage, **this**.dbs, ae);

ed.doExceptionDialog();

op = **null**;

}

}

**return** op;

}

AccessOperation :

package model.orm;

import java.sql.CallableStatement;

import java.sql.Connection;

import java.sql.Date;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.Calendar;

import model.data.Operation;

import model.orm.exception.DataAccessException;

import model.orm.exception.DatabaseConnexionException;

import model.orm.exception.ManagementRuleViolation;

import model.orm.exception.Order;

import model.orm.exception.RowNotFoundOrTooManyRowsException;

import model.orm.exception.Table;

public class AccessOperation {

public AccessOperation() {

}

/\*\*

\* Recherche de toutes les opérations d'un compte.

\*

\* @param idNumCompte id du compte dont on cherche toutes les opérations

\* @return Toutes les opérations du compte, liste vide si pas d'opération

\* @throws DataAccessException

\* @throws DatabaseConnexionException

\*/

public ArrayList<Operation> getOperations(int idNumCompte) throws DataAccessException, DatabaseConnexionException {

ArrayList<Operation> alResult = new ArrayList<>();

try {

Connection con = LogToDatabase.getConnexion();

String query = "SELECT \* FROM Operation where idNumCompte = ?";

query += " ORDER BY dateOp";

PreparedStatement pst = con.prepareStatement(query);

pst.setInt(1, idNumCompte);

ResultSet rs = pst.executeQuery();

while (rs.next()) {

int idOperation = rs.getInt("idOperation");

double montant = rs.getDouble("montant");

Date dateOp = rs.getDate("dateOp");

Date dateValeur = rs.getDate("dateValeur");

int idNumCompteTrouve = rs.getInt("idNumCompte");

String idTypeOp = rs.getString("idTypeOp");

alResult.add(new Operation(idOperation, montant, dateOp, dateValeur, idNumCompteTrouve, idTypeOp));

}

rs.close();

pst.close();

return alResult;

} catch (SQLException e) {

throw new DataAccessException(Table.Operation, Order.SELECT, "Erreur accès", e);

}

}

/\*\*

\* Recherche d'une opération par son id.

\*

\* @param idOperation id de l'opération recherchée (clé primaire)

\* @return une Operation ou null si non trouvé

\* @throws RowNotFoundOrTooManyRowsException

\* @throws DataAccessException

\* @throws DatabaseConnexionException

\*/

public Operation getOperation(int idOperation)

throws RowNotFoundOrTooManyRowsException, DataAccessException, DatabaseConnexionException {

Operation operationTrouvee;

try {

Connection con = LogToDatabase.getConnexion();

String query = "SELECT \* FROM Operation where" + " idOperation = ?";

PreparedStatement pst = con.prepareStatement(query);

pst.setInt(1, idOperation);

ResultSet rs = pst.executeQuery();

if (rs.next()) {

int idOperationTrouve = rs.getInt("idOperation");

double montant = rs.getDouble("montant");

Date dateOp = rs.getDate("dateOp");

Date dateValeur = rs.getDate("dateValeur");

int idNumCompteTrouve = rs.getInt("idNumCompte");

String idTypeOp = rs.getString("idTypeOp");

operationTrouvee = new Operation(idOperationTrouve, montant, dateOp, dateValeur, idNumCompteTrouve,

idTypeOp);

} else {

rs.close();

pst.close();

return null;

}

if (rs.next()) {

rs.close();

pst.close();

throw new RowNotFoundOrTooManyRowsException(Table.Operation, Order.SELECT,

"Recherche anormale (en trouve au moins 2)", null, 2);

}

rs.close();

pst.close();

return operationTrouvee;

} catch (SQLException e) {

throw new DataAccessException(Table.Operation, Order.SELECT, "Erreur accès", e);

}

}

/\*\*

\* Enregistrement d'un débit.

\*

\* Se fait par procédure stockée : - Vérifie que le débitAutorisé n'est pas

\* dépassé - Enregistre l'opération - Met à jour le solde du compte.

\*

\* @param idNumCompte compte débité

\* @param montant montant débité

\* @param typeOp libellé de l'opération effectuée (cf TypeOperation)

\* @throws RowNotFoundOrTooManyRowsException

\* @throws DataAccessException

\* @throws DatabaseConnexionException

\* @throws ManagementRuleViolation

\*/

public void insertDebit(int idNumCompte, double montant, String typeOp)

throws DatabaseConnexionException, ManagementRuleViolation, DataAccessException {

try {

Connection con = LogToDatabase.getConnexion();

CallableStatement call;

String q = "{call Debiter (?, ?, ?, ?)}";

// les ? correspondent aux paramètres : cf. déf procédure (4 paramètres)

call = con.prepareCall(q);

// Paramètres in

call.setInt(1, idNumCompte);

// 1 -> valeur du premier paramètre, cf. déf procédure

call.setDouble(2, montant);

call.setString(3, typeOp);

// Paramètres out

call.registerOutParameter(4, java.sql.Types.INTEGER);

// 4 type du quatrième paramètre qui est déclaré en OUT, cf. déf procédure

call.execute();

int res = call.getInt(4);

if (res != 0) { // Erreur applicative

throw new ManagementRuleViolation(Table.Operation, Order.INSERT,

"Erreur de règle de gestion : découvert autorisé dépassé", null);

}

} catch (SQLException e) {

throw new DataAccessException(Table.Operation, Order.INSERT, "Erreur accès", e);

}

}

/\*\*

\* Enregistrement d'un débit.

\*

\* Se fait par procédure stockée : - Vérifie que le débitAutorisé n'est pas

\* dépassé - Enregistre l'opération - Met à jour le solde du compte.

\*

\* @param idNumCompte compte débité

\* @param montant montant débité

\* @param typeOp libellé de l'opération effectuée (cf TypeOperation)

\* @throws RowNotFoundOrTooManyRowsException

\* @throws DataAccessException

\* @throws DatabaseConnexionException

\* @throws ManagementRuleViolation

\*/

public void insertCredit(int idNumCompte, double montant, String typeOp)

throws DatabaseConnexionException, ManagementRuleViolation, DataAccessException {

try {

Connection con = LogToDatabase.getConnexion();

CallableStatement call;

String q = "{call Debiter (?, ?, ?, ?)}";

// les ? correspondent aux paramètres : cf. déf procédure (4 paramètres)

call = con.prepareCall(q);

// Paramètres in

call.setInt(1, idNumCompte);

// 1 -> valeur du premier paramètre, cf. déf procédure

call.setDouble(2, -montant);

call.setString(3, typeOp);

// Paramètres out

call.registerOutParameter(4, java.sql.Types.INTEGER);

// 4 type du quatrième paramètre qui est déclaré en OUT, cf. déf procédure

call.execute();

int res = call.getInt(4);

if (res != 0) { // Erreur applicative

throw new ManagementRuleViolation(Table.Operation, Order.INSERT,

"Erreur de règle de gestion : découvert autorisé dépassé", null);

}

} catch (SQLException e) {

throw new DataAccessException(Table.Operation, Order.INSERT, "Erreur accès", e);

}

}

/\*\*

\* Enregistrement d'un débit.

\*

\* Se fait par procédure stockée : - Vérifie que le débitAutorisé n'est pas

\* dépassé - Enregistre l'opération - Met à jour le solde du compte.

\*

\* @param idNumCompte compte débité

\* @param montant montant débité

\* @param typeOp libellé de l'opération effectuée (cf TypeOperation)

\* @throws RowNotFoundOrTooManyRowsException

\* @throws DataAccessException

\* @throws DatabaseConnexionException

\* @throws ManagementRuleViolation

\*/

public void insertVirement(int idNumCompte, double montant, String typeOp, int idCompteVise)

throws DatabaseConnexionException, ManagementRuleViolation, DataAccessException {

try {

Connection con = LogToDatabase.getConnexion();

CallableStatement call;

String q = "{call Virer (?, ?, ?, ?)}";

// les ? correspondent aux paramètres : cf. déf procédure (4 paramètres)

call = con.prepareCall(q);

// Paramètres in

call.setInt(1, idNumCompte);

// 1 -> valeur du premier paramètre, cf. déf procédure

call.setDouble(2, idCompteVise);

call.setDouble(3, montant);

// Paramètres out

call.registerOutParameter(4, java.sql.Types.INTEGER);

// 4 type du quatrième paramètre qui est déclaré en OUT, cf. déf procédure

call.execute();

int res = call.getInt(4);

if (res != 0) { // Erreur applicative

throw new ManagementRuleViolation(Table.Operation, Order.INSERT,

"Erreur de règle de gestion : découvert autorisé dépassé", null);

}

} catch (SQLException e) {

throw new DataAccessException(Table.Operation, Order.INSERT, "Erreur accès", e);

}

}

/\*\*

\* Fonction utilitaire qui retourne un ordre sql "to\_date" pour mettre une date

\* dans une requête sql

\*

\* @param d Date (java.sql) à transformer

\* @return Une chaine : TO\_DATE ('j/m/a', 'DD/MM/YYYY') 'j/m/a' : jour mois an

\* de d ex : TO\_DATE ('25/01/2019', 'DD/MM/YYYY')

\*/

private String dateToString(Date d) {

String sd;

Calendar cal;

cal = Calendar.getInstance();

cal.setTime(d);

sd = "" + cal.get(Calendar.DAY\_OF\_MONTH) + "/" + (cal.get(Calendar.MONTH) + 1) + "/" + cal.get(Calendar.YEAR);

sd = "TO\_DATE( '" + sd + "' , 'DD/MM/YYYY')";

return sd;

}

}