Correction [IHM] 2025

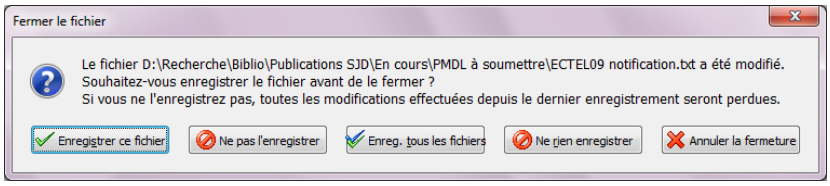
EX1 (12P):

1 Qu'est-ce que la Loi de Hick ? 1

Le temps nécessaire pour prendre une décision dépend

- **Du nombre**
- **Et de la complexité**
- **Des options proposes**

2 Donnez un exemple concret illustrant la loi de Hick dans un contexte d'interface utilisateur ?1



3 Quelles sont les limites des méthodes de conception gène logiciel dans la conception d'IHM ?3

- **Implication limitée des utilisateurs**
- **Méthodes centrées système (garantie fonctionnelle) au détriment des utilisateurs**
- **Évaluation tardive**

4 Quelles sont les informations collectées lors de la conception d'une IHM ?1,5

- **Sur les utilisateurs (e.g., pour construire des personas)**
- **Sur les tâches (e.g., enchaînement des actions, vocabulaire métier)**
- **Sur les interfaces, notamment en évaluation (e.g., idées, points forts/faibles)**

5 Quel est l'avantage principal d'une approche déclarative pour construire des interfaces utilisateur ?1,5

- **Une séparation nette entre le code métier et l'interface graphique utilisateur.**

6 Quels sont les outils disponibles pour aider au développement d'interfaces graphiques déclaratives(javafx) ?0,5

- **SceneBuilder**

7 Donnez un exemple simple d'interface graphique déclarative pour FXML ?2

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

<?import javafx.geometry.*?>
<?import javafx.scene.text.*?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>

<BorderPane  . . .  >

  . . .

</BorderPane>
```

8 Expliquer les concepts suivants : Croquis, Maquette, Prototype ?1,5

-Croquis (sketch) = aperçu global de l'interface (idée générale)

-Maquette (mockup, wireframe) = interface détaillée (sans interaction)

-Prototype = version incomplète d'une interface (avec interactions)

Ex2 :

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class CalculatorFX extends Application {
   private TextField operand1Field;
   private TextField operand2Field;
   private TextField resultField;
   @Override
   public void start(Stage primaryStage) {
      Label operand1Label = new Label("First operand:");
      Label operand2Label = new Label("Second operand:");
      Label resultLabel = new Label("Result:");
      operand1Field = new TextField();
      operand2Field = new TextField();
      resultField = new TextField();
      resultField.setEditable(false);
      Button addButton = new Button("Add");
      Button subtractButton = new Button("Subtract");
      Button multiplyButton = new Button("Multiply");
      Button divideButton = new Button("Divide");
      addButton.setOnAction(e -> calculate('+'));
      subtractButton.setOnAction(e -> calculate('-'));
      multiplyButton.setOnAction(e -> calculate('*'));
      divideButton.setOnAction(e -> calculate('/'));
      HBox buttonBox = new HBox(10, addButton, subtractButton, multiplyButton, divideButton);
      buttonBox.setAlignment(Pos.CENTER);
      GridPane grid = new GridPane();
      grid.setPadding(new Insets(10));
      grid.setHgap(5);    grid.setVgap(5);
      grid.add(operand1Label, 0, 0);
      grid.add(operand1Field, 1, 0);
      grid.add(operand2Label, 0, 1);
      grid.add(operand2Field, 1, 1);
      grid.add(resultLabel, 0, 2);
      grid.add(resultField, 1, 2);
      VBox mainLayout = new VBox(10, grid, buttonBox);
      mainLayout.setPadding(new Insets(10));
      primaryStage.setTitle("Calculator");
      Scene scene = new Scene(mainLayout);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   private void calculate(char operator) {
      try {
         double operand1 = Double.parseDouble(operand1Field.getText());
         double operand2 = Double.parseDouble(operand2Field.getText());
         double result = 0;
         switch (operator) {
            case '+':   result = operand1 + operand2;   break;
            case '-':   result = operand1 - operand2;    break;
            case '*':    result = operand1 * operand2;    break;
            case '/':   if (operand2 == 0) { resultField.setText("Division by zero!");  return;  }
                        result = operand1 / operand2;    break;
         }
         resultField.setText(String.valueOf(result));
      } catch (NumberFormatException e) {  resultField.setText("Invalid input!"); }
   }
   public static void main(String[] args) {  launch(args);   }
}
```