**Antisèche TD Noté - R3.04 C#**

**1. Binding WPF**

* **Ne pas nommer les TextBox (pas de x:Name).**
* **Utiliser le DataContext** pour le binding :

public partial class MainWindow : Window, INotifyPropertyChanged {

public event PropertyChangedEventHandler PropertyChanged;

public double Nb1 { get; set; }

private double resultat;

public double Resultat {

get { return resultat; }

set { resultat = value; OnPropertyChanged("Resultat"); }

}

protected void OnPropertyChanged(string name) {

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));

}

}

* **Mode de binding :**

<TextBox Text="{Binding Path=Nb1, Mode=TwoWay}"/>

**2. Architecture et Séparation du Code**

* **Créer une bibliothèque de classes "Calculatrice"** :

Classe statique avec les méthodes :

public static class Calcul {

public static double Addition(double a, double b) => a + b;

public static double Soustraction(double a, double b) => a - b;

public static double Multiplication(double a, double b) => a \* b;

public static double Division(double a, double b) {

if (b == 0) throw new DivideByZeroException("Erreur division par zéro");

return a / b;

}

}

* + **Gérer les exceptions avec try-catch** :

try {

double result = Calcul.Division(Nb1, Nb2);

} catch (DivideByZeroException ex) {

MessageBox.Show(ex.Message);

}

**3. Debugging Visual Studio**

* **Breakpoints (F9)** : Arrêter l'exécution à une ligne donnée.
* **Step Into (F11)** : Entrer dans une fonction.
* **Step Over (F10)** : Exécuter une fonction sans y entrer.
* **Step Out (Shift+F11)** : Sortir d'une fonction en cours.
* **Espions (Watch)** : Ajouter une variable à la surveillance.
* **Breakpoints conditionnels** :

if (n == 12) Debugger.Break();

* **Mode Debug vs Release** :

#if DEBUG

Console.WriteLine("Mode Debug");

#endif

**4. Manipulation JSON**

* **Lecture et écriture d'un fichier JSON** :

public static class Service {

private static readonly string filePath = "Data/Devises.json";

public static List<Devise> LoadDevises() {

if (!File.Exists(filePath)) return new List<Devise>();

string json = File.ReadAllText(filePath);

return JsonSerializer.Deserialize<List<Devise>>(json) ?? new List<Devise>();

}

}

* **Classe Devise pour la désérialisation** :

public class Devise {

public string NomDevise { get; set; }

public double Taux { get; set; }

}

**5. Tests Unitaires**

* **Framework MSTest** :

[TestClass]

public class CalculTests {

[TestMethod]

public void AdditionTest() {

Assert.AreEqual(3.0, Calcul.Addition(1, 2));

}

}

* **Test d'exception** :

[TestMethod]

[ExpectedException(typeof(DivideByZeroException))]

public void DivisionTest\_Exception() {

Calcul.Division(1, 0);

}

* **Assertions utiles** :

Assert.IsTrue(condition);

Assert.IsFalse(condition);

Assert.IsNull(objet);

Assert.IsNotNull(objet);

**6. Accès aux Données avec PostgreSQL**

* **Connexion et exécution d'une requête** :

using (var conn = new NpgsqlConnection("Host=localhost;Database=BDComptesBancaires;Username=user;Password=pass")) {

conn.Open();

using (var cmd = new NpgsqlCommand("SELECT \* FROM Compte", conn)) {

using (var reader = cmd.ExecuteReader()) {

while (reader.Read()) {

Console.WriteLine(reader["id"] + " - " + reader["solde"]);

}

}

}

}

* **Exécution d'une procédure stockée** :

using (var cmd = new NpgsqlCommand("sp\_virement\_append", conn)) {

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("pidcomptedebit", 1);

cmd.Parameters.AddWithValue("pidcomptecredit", 2);

cmd.Parameters.AddWithValue("pmontant", 100);

cmd.ExecuteNonQuery();

}

**7. Gestion des Transactions SQL**

* **Exécution d'une transaction pour un virement** :

BEGIN;

UPDATE compte SET solde = solde - 100 WHERE idcompte = 1;

UPDATE compte SET solde = solde + 100 WHERE idcompte = 2;

INSERT INTO virement (idcomptedebit, idcomptecredit, montant) VALUES (1, 2, 100);

COMMIT;

**8. Bonnes Pratiques**

* **Utiliser des procédures stockées pour éviter l'injection SQL.**
* **Ne jamais faire confiance aux entrées utilisateurs (toujours valider et nettoyer).**
* **Séparer les couches :**
  + UI (Interface utilisateur - WPF)
  + Business Layer (Métier)
  + Data Layer (Accès BD)
* **Écrire des tests unitaires systématiquement.**
* **Utiliser le versionnement Git.**
* **Respecter les conventions de nommage et structuration du code.**
* **Commenter et documenter le code avec XMLDoc et Doxygen.**

Cette antisèche regroupe les éléments **essentiels et pratiques** pour réussir le TD noté rapidement et efficacement.