# Invoice Master
# Extraits de code commentés

## 1. Transactions Business Logic Layer (TransactionsBLL)

```
// Classe représentant une transaction générique (vente ou achat)
// Utilisation : instancier et remplir cette entité avant de la passer à TransactionDAL.InsertTransaction().
3 references
internal class TransactionsBLL
{
    // Identifiant unique de la transaction
    0 references
    public int id { get; set; }
    // Type de transaction : "ACHATS" ou "VENTES"
    2 references
    public string type { get; set; }
    // Référence au client ou fournisseur
    2 references
    public int dea_cust_id { get; set; }
    // Montant total avant TVA et remise
    2 references
    public decimal grand_total { get; set; }
    // Date et heure de la transaction
    2 references
    public DateTime transaction_date { get; set; }
    // Taux de TVA appliqué
    2 references
    public decimal vat { get; set; }
    // Pourcentage de remise appliqué
    2 references
    public decimal discount { get; set; }
    // Identifiant de l'utilisateur ayant créé la transaction
    2 references
    public int added_by { get; set; }
    // Détails des lignes de la transaction (produit, quantité, prix)
    1 reference
    public DataTable TransactionDetails { get; set; }
}
```

## 2. Détails de Transaction Business Logic Layer (TransactionDetailBLL)

```
// Classe représentant une ligne de détail dans une transaction
// Utilisation : remplir pour chaque ligne et passer à TransactionDetailDAL.InsertTransactionDetail().
3 references
internal class TransactionDetailBLL
{
    // Identifiant unique du détail
    0 references
    public int id { get; set; }
    // Référence au produit concerné
    4 references
    public int product_id { get; set; }
    // Tarif unitaire du produit
    2 references
    public decimal rate { get; set; }
    // Quantité vendue ou achetée
    4 references
    public decimal qty { get; set; }
    // Total par ligne (rate * qty)
    2 references
    public decimal total { get; set; }
    // Identique à dea_cust_id de la transaction parente
    2 references
    public int dea_cust_id { get; set; }
    // Date d'ajout du détail
    2 references
    public DateTime added_date { get; set; }
    // Utilisateur ayant ajouté ce détail
    2 references
    public int added_by { get; set; }
}
```

## 3. Interface Transactions (frmTransactions)

```
5 references
public partial class frmTransactions : Form
{
    // Constructor
    1 reference
    public frmTransactions()
    {
        InitializeComponent();
    }

    1 reference
    private void pictureBoxClose_Click(object sender, EventArgs e)...

    // Instance de la classe TransactionDAL
    TransactionDAL tdal = new TransactionDAL();

    // Au chargement, récupère et affiche toutes les transactions
    1 reference
    private void frmTransactions_Load(object sender, EventArgs e)
    {
        DataTable dt = tdal.DisplayAllTransactions();
        dgvTransactions.DataSource = dt;
    }

    // Au clic, récupère et affiche toutes les transactions
    1 reference
    private void btnShowAll_Click(object sender, EventArgs e)
    {
        DataTable dt = tdal.DisplayAllTransactions();
        dgvTransactions.DataSource = dt;
    }

    // Au changement du type, filtre dynamiquement les transactions par type via DisplayTransactionByType
    1 reference
    private void cmbTransactionType_SelectedIndexChanged(object sender, EventArgs e)
    {
        //Récupère le type de transaction sélectionné
        string type = cmbTransactionType.Text;
        //Récupère et affiche les transactions par type
        DataTable dt = tdal.DisplayTransactionByType(type);
        dgvTransactions.DataSource = dt;
    }
```

## 4. Interface Achats et Ventes (frmPurchasesAndSales)

```
7 references
public partial class frmPurchasesAndSales : Form
{
    // Constructor
    2 references
    public frmPurchasesAndSales()
    {
        InitializeComponent();
    }

    1 reference
    private void pictureBoxClose_Click(object sender, EventArgs e)...

    // Instances des classes DAL
    DeaCustDAL dcDAL = new DeaCustDAL();
    ProductsDAL pDAL = new ProductsDAL();
    UserDAL uDAL = new UserDAL();
    TransactionDAL tDAL = new TransactionDAL();
    TransactionDetailDAL tdDAL = new TransactionDetailDAL();
    // Création d'une instance de DataTable pour stocker les transactions
    DataTable transactionDT = new DataTable();

    // Au chargement, récupère et affiche le type de transaction
    // et initialise les colonnes de la DataTable
    1 reference
    private void frmPurchasesAndSales_Load(object sender, EventArgs e)
    {
        // Récupère le type de transaction (achat ou vente) depuis le formulaire du tableau de bord
        string type = frmUserDashboard.transactionType;
        // Affiche le type de transaction dans le label de la fenêtre
        lblTop.Text = type;

        // Initialise les colonnes de la DataTable pour stocker les transactions
        transactionDT.Columns.Add("Nom du produit");
        transactionDT.Columns.Add("Prix");
        transactionDT.Columns.Add("Quantité");
        transactionDT.Columns.Add("Total");

    }
```

```
// Affiche les détails du fournisseur ou du client à la saisie dans la barre de recherche
1 reference
private void txtDeaCustSearch_TextChanged(object sender, EventArgs e)
{
    // Récupère le texte de recherche
    string keywords = txtDeaCustSearch.Text;

    // Si la barre de recherche est vide, efface les champs de texte
    if (keywords == "")
    {
        txtDeaCustName.Text = "";
        txtEmail.Text = "";
        txtContact.Text = "";
        txtAddress.Text = "";
        return;
    }

    // Instance la classe DeaCustBLL pour rechercher le fournisseur ou le client
    DeaCustBLL dc = dcDAL.SearchDealerCustomerTransaction(keywords);

    // Affiche les détails du fournisseur ou du client dans les champs de texte
    txtDeaCustName.Text = dc.name;
    txtEmail.Text = dc.email;
    txtContact.Text = dc.contact;
    txtAddress.Text = dc.address;
}
```

```
// Affiche les détails du produit à la saisie dans la barre de recherche
1 reference
private void txtProductSearch_TextChanged(object sender, EventArgs e)
{
    // Récupère le texte de recherche
    string keywords = txtProductSearch.Text;

    // Si la barre de recherche est vide, efface les champs de texte
    if (keywords == "")
    {
        txtProductName.Text = "";
        txtInventory.Text = "";
        txtRate.Text = "";
        txtQty.Text = "";
        return;
    }

    // Instance la classe ProductsBLL pour rechercher le produit
    ProductsBLL p = pDAL.SearchProductTransaction(keywords);

    // Affiche les détails du produit dans les champs de texte
    txtProductName.Text = p.name;
    txtInventory.Text = p.qty.ToString();
    txtRate.Text = p.rate.ToString();
}
```

```
// Ajoute le produit sélectionné à la liste des produits ajoutés
1 reference
private void btnAdd_Click(object sender, EventArgs e)
{
    // Récupère les valeurs des champs de texte
    string productName = txtProductName.Text;

    // Vérifie si les montants saisis sont valides, retourne un message d'erreur sinon
    if (!decimal.TryParse(txtRate.Text, out decimal rate))
    {
        MessageBox.Show("Prix invalide : entrer un nombre.", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }
    if (!decimal.TryParse(txtQty.Text, out decimal qty))
    {
        MessageBox.Show("Quantité invalide : entrer un nombre.", "Erreur", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    // Calcule le total pour la ligne
    decimal total = rate * qty;

    // Calcule le sous-total pour la transaction entière
    decimal subTotal;
    decimal.TryParse(txtSubTotal.Text, out subTotal);
    subTotal = subTotal + total;

    // Si le nom du produit est vide, affiche un message d'erreur
    if (productName == "")
    {
        MessageBox.Show("Sélectionner un produit et réessayer.");
    }
    // Sinon , ajoute le produit à la liste des produits ajoutés et vide les champs de texte
    else
    {
        // Ajoute le produit à la DataTable et l'affiche dans le DataGridView
        transactionDT.Rows.Add(productName, rate, qty, total);
        dgvAddedProducts.DataSource = transactionDT;

        // Affiche les données dans les détails de calcul
        txtSubTotal.Text = subTotal.ToString();

        // Vide les champs de texte
        txtProductSearch.Text = "";
        txtProductName.Text = "";
        txtInventory.Text = "0";
        txtRate.Text = "0.00";
        txtQty.Text = "0";
    }
}
```

```
// Calcul le total en fonction de la remise
1 reference
private void txtDiscount_TextChanged(object sender, EventArgs e)
{
    // Récupère la valeur de la remise
    string value = txtDiscount.Text;

    // Si la remise est vide, affiche un message d'erreur
    if (value == "")
    {
        MessageBox.Show("Saisir une remise (0 si pas de remise).");
    }
    // Sinon, calcule le total en fonction de la remise
    else
    {
        // Récupère le montant de la remise
        decimal discount;
        decimal.TryParse(txtDiscount.Text, out discount);
        // Récupère le sous-total
        decimal subTotal;
        decimal.TryParse(txtSubTotal.Text, out subTotal);

        // Calcul le total après remise
        decimal grandTotal = ((100 - discount) / 100) * subTotal;

        // Affiche le total dans le champ de texte
        txtGrandTotal.Text = grandTotal.ToString();
    }
}
```

```
// Calcul le total en fonction de la TVA
1 reference
private void txtVAT_TextChanged(object sender, EventArgs e)
{
    // Si la TVA est vide, affiche un message d'erreur
    string check = txtGrandTotal.Text;
    if (check == "")
    {
        MessageBox.Show("Calculer d'abord la remise pour définir le total");
    }
    // Sinon, calcule le total en fonction de la TVA
    else
    {
        // Récupère l'ancien total et la TVA
        decimal previousGT = decimal.Parse(txtGrandTotal.Text);
        decimal vat;
        decimal.TryParse(txtVAT.Text, out vat);

        //Calcul le total avec la TVA
        decimal grandTotalWithVAT = ((100 + vat ) / 100 ) * previousGT;

        // Affiche le total dans le champ de texte
        txtGrandTotal.Text = grandTotalWithVAT.ToString();
    }
}

// Calcul le montant à rendre au client
1 reference
private void txtPaidAmount_TextChanged(object sender, EventArgs e)
{
    // Récupère le montant total
    decimal grandTotal = decimal.Parse(txtGrandTotal.Text);
    // Récupère le montant payé
    decimal paidAmount;
    decimal.TryParse(txtPaidAmount.Text, out paidAmount);

    // Calcul le montant à rendre et l'affiche dans le champ de texte
    decimal returnAmount = paidAmount - grandTotal;
    txtReturnAmount.Text = returnAmount.ToString();
}
```

```
// Enregistre la transaction
1 reference
private void btnSave_Click(object sender, EventArgs e)
{
    // Instance la classe TransactionsBLL pour enregistrer la transaction
    TransactionsBLL transaction = new TransactionsBLL();

    // Récupère le type de transaction depuis le label
    transaction.type = lblTop.Text;

    // Récupère le nom du fournisseur ou du client
    string deaCustName = txtDeaCustName.Text;

    // Instance la classe DeaCustBLL pour récupérer l'ID du fournisseur ou du client
    DeaCustBLL dc = dcDAL.GetDeaCustIDFromName(deaCustName);

    // Attribue les valeurs aux propriétés de la transaction
    transaction.dea_cust_id = dc.id;
    transaction.grand_total = Math.Round(decimal.Parse(txtGrandTotal.Text), 2);
    transaction.transaction_date = DateTime.Now;
    transaction.vat = decimal.TryParse(txtVAT.Text.Trim().Replace('.', ','), out decimal v) ? v : 0m;
    transaction.discount = decimal.Parse(txtDiscount.Text);

    // Récupère le nom d'utilisateur connecté
    string username = frmLogin.loggedIn;

    // Instance la classe UserBLL pour récupérer l'ID de l'utilisateur
    UserBLL u = uDAL.GetIDFromUsername(username);

    // Attribue l'ID de l'utilisateur à la transaction
    transaction.added_by = u.id;

    // Attribue les valeurs à la DataTable de la transaction
    transaction.TransactionDetails = transactionDT;

    // Vérifie si les données ont été inserées avec succès
    bool isSuccess = false;

    // Utilise une transaction pour garantir l'intégrité des données
    using (TransactionScope scope = new TransactionScope())
    {
        // Instancie l'ID de la transaction à -1
        int transactionID = -1;

        // Variable booléenne pour vérifier si la transaction a été insérée
        bool w = tDAL.InsertTransaction(transaction, out transactionID);

        //Boucle pour insérer les détails de la transaction
        for (int i = 0; i < transactionDT.Rows.Count; i++)
        {
            // Instancie la classe TransactionDetailBLL pour insérer les détails de la transaction
            TransactionDetailBLL transactionDetail = new TransactionDetailBLL();
            string ProductName = transactionDT.Rows[i][0].ToString();
            // Instancie la classe ProductsBLL pour récupérer l'ID du produit
            ProductsBLL p = pDAL.GetProductIDFromName(ProductName);

            // Attribue les valeurs aux propriétés des détails de la transaction
            transactionDetail.product_id = p.id;
            transactionDetail.rate = decimal.Parse(transactionDT.Rows[i][1].ToString());
            transactionDetail.qty = decimal.Parse(transactionDT.Rows[i][2].ToString());
            transactionDetail.total = Math.Round(decimal.Parse(transactionDT.Rows[i][3].ToString()), 2);
            transactionDetail.dea_cust_id = dc.id;
            transactionDetail.added_date = DateTime.Now;
            transactionDetail.added_by = u.id;

            // Récupère le type de transaction depuis le label
            string transactionType = lblTop.Text;

            // Si la transaction est un achat, augmente la quantité du produit
            bool x = false;
            if (transactionType == "ACHATS")
            {
                x = pDAL.IncreaseProduct(transactionDetail.product_id, transactionDetail.qty);
            }
            // Sinon si la transaction est une vente, diminue la quantité du produit
            else if (transactionType == "VENTES")
            {
                x = pDAL.DecreaseProduct(transactionDetail.product_id, transactionDetail.qty);
            }

            // Insert les détails de la transaction dans la base de données
            bool y = tdDAL.InsertTransactionDetail(transactionDetail, out _);
            isSuccess = w && x && y;
        }

        // Si la transaction a été insérée avec succès, complète la transaction
        if (isSuccess == true)
        {
            scope.Complete();
        }
        // Sinon, affiche un message d'erreur
        else
        {
            MessageBox.Show("Échec, la transaction n'a pas abouti.");
        }
    }
}
```

## 5. Data Access Layer Transactions (TransactionDAL)

```
4 references
internal class TransactionDAL
{
    // Chaine de caractère de connexion à la base de données
    static string myconnstrng = ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;

    #region Insert Transaction Method
    // Insert les données de la transaction dans la base de données
    1 reference
    public bool InsertTransaction(TransactionsBLL t, out int transactionID)
    {

        bool isSuccess = false; // Vérifie si la transaction a été insérée avec succès

        SqlConnection conn = new SqlConnection(myconnstrng); // Connexion à la base de données

        transactionID = -1; //Attribue une valeur par défaut à transactionID

        try
        {
            // Requête SQL pour insérer les données dans la table tbl_transactions
            string sql = "INSERT INTO tbl_transactions " +
                "(type, dea_cust_id, grand_total, transaction_date, vat, discount, added_by) " +
                "VALUES (@type, @dea_cust_id, @grand_total, @transaction_date, @vat, @discount, @added_by);" +
                "SELECT @@IDENTITY;";

            SqlCommand cmd = new SqlCommand(sql, conn); // Commande Sql pour exécuter la requête SQL

            // Ajoute les paramètres à la commande Sql
            cmd.Parameters.AddWithValue("@type", t.type);
            cmd.Parameters.AddWithValue("@dea_cust_id", t.dea_cust_id);
            cmd.Parameters.AddWithValue("@grand_total", t.grand_total);
            cmd.Parameters.AddWithValue("@transaction_date", t.transaction_date);
            cmd.Parameters.AddWithValue("@vat", t.vat);
            cmd.Parameters.AddWithValue("@discount", t.discount);
            cmd.Parameters.AddWithValue("@added_by", t.added_by);

            conn.Open(); // Ouvre la connexion à la base de données

            object o = cmd.ExecuteScalar(); // Exécute la commande et récupère l'ID de la transaction

            // Si l'ID de la transaction n'est pas nul, la transaction a été insérée avec succès
            if (o != null)
            {
                transactionID = int.Parse(o.ToString()); // L'ID de la transaction est converti en entier
                isSuccess = true;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message); // Affiche un message d'erreur si une exception se produit
        }
        finally
        {
            conn.Close(); // Ferme la connexion à la base de données
        }

        return isSuccess; // Retourne true si la transaction a été insérée avec succès, sinon false
    }
    #endregion
```

```
#region Display all the Transaction
// Affiche toutes les transactions de la base de données
2 references
public DataTable DisplayAllTransactions()
{
    SqlConnection conn = new SqlConnection(myconnstrng); // Connexion à la base de données

    DataTable dt = new DataTable(); // DataTable pour stocker les données

    try
    {
        // Requête SQL pour sélectionner toutes les données de la table tbl_transactions
        string sql = "SELECT * FROM tbl_transactions";

        SqlCommand cmd = new SqlCommand(sql, conn); // Commande Sql pour exécuter la requête SQL

        SqlDataAdapter adapter = new SqlDataAdapter(cmd); // SqlDataAdapter pour remplir le DataTable avec les données

        conn.Open(); // Ouvre la connexion à la base de données

        adapter.Fill(dt); // Remplit le DataTable avec les données de la base de données
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message); // Affiche un message d'erreur si une exception se produit
    }
    finally
    {
        conn.Close(); // Ferme la connexion à la base de données
    }
    return dt; // Retourne le DataTable avec les données
}
#endregion
```

```
#region Display Transaction by Type
// Affiche les transactions par type
1 reference
public DataTable DisplayTransactionByType(string type)
{
    SqlConnection conn = new SqlConnection(myconnstrng); // Connexion à la base de données

    DataTable dt = new DataTable(); // DataTable pour stocker les données

    try
    {
        // Requête SQL pour sélectionner les transactions par type
        string sql = "SELECT * FROM tbl_transactions WHERE type=@t";

        SqlCommand cmd = new SqlCommand(sql, conn); // Commande Sql pour exécuter la requête SQL

        cmd.Parameters.AddWithValue("@t", type); // Ajoute le paramètre @t à la commande Sql

        SqlDataAdapter adapter = new SqlDataAdapter(cmd); // SqlDataAdapter pour remplir le DataTable avec les données

        conn.Open(); // Ouvre la connexion à la base de données

        adapter.Fill(dt); // Remplit le DataTable avec les données de la base de données
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message); // Affiche un message d'erreur si une exception se produit
    }
    finally
    {
        conn.Close(); // Ferme la connexion à la base de données
    }
    return dt;
}
#endregion
```

## 6. Data Access Layer Détails (TransactionDetailDAL)

```
2 references
internal class TransactionDetailDAL
{
    // Chaine de caracteres de connexion à la base de données
    static string myconnstrng = ConfigurationManager.ConnectionStrings["connstrng"].ConnectionString;

    #region Insert Transaction Detail Method
    // Insert les détails de la transaction dans la base de données
    1 reference
    public bool InsertTransactionDetail(TransactionDetailBLL td, out int transactionID)
    {
        bool isSuccess = false; // Variable booléenne pour indiquer si l'insertion a réussi

        SqlConnection conn = new SqlConnection(myconnstrng); // Connexion à la base de données

        transactionID = -1; // Variable pour stocker l'ID de la transaction insérée

        try
        {
            // Requête SQL pour insérer les détails de la transaction
            string sql = "INSERT INTO tbl_transaction_detail " +
                "(product_id, rate, qty, total, dea_cust_id, added_date, added_by) " +
                "VALUES (@product_id, @rate, @qty, @total, @dea_cust_id, @added_date, @added_by)";

            SqlCommand cmd = new SqlCommand(sql, conn); // Commande SQL pour éxécuter la requête

            // Ajout des paramètres à la commande SQL
            cmd.Parameters.AddWithValue("@product_id", td.product_id);
            cmd.Parameters.AddWithValue("@rate", td.rate);
            cmd.Parameters.AddWithValue("@qty", td.qty);
            cmd.Parameters.AddWithValue("@total", td.total);
            cmd.Parameters.AddWithValue("@dea_cust_id", td.dea_cust_id);
            cmd.Parameters.AddWithValue("@added_date", td.added_date);
            cmd.Parameters.AddWithValue("@added_by", td.added_by);

            conn.Open(); // Ouverture de la connexion à la base de données

            int rows = cmd.ExecuteNonQuery(); // Variable pour stocker le nombre de lignes affectées par la requête
            if (rows > 0)
                isSuccess = true; // Si au moins une ligne a été affectée, l'insertion a réussi
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message); // Affichage d'un message d'erreur en cas d'exception
        }
        finally
        {
            conn.Close(); // Fermeture de la connexion à la base de données
        }

        return isSuccess; // Retourne true si l'insertion a réussi, sinon false
    }
    #endregion
}
```