using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MyDrive

{

public partial class frmEstimate : Form

{

// Variable

private decimal exchangeRate;

private decimal priceCurrentMaterial;

private string materialName;

private decimal foundations;

private string foundationName;

private decimal[] pricesOfMaterials;

public frmEstimate()

{

InitializeComponent();

pricesOfMaterials = new decimal[4];

foundationName = "Standard";

// set for easy start

//txtExchangeRate.Text = (1.45).ToString();

//txtBrick.Text = (35.75).ToString();

//txtConcrete.Text = (25.50).ToString();

//txtTarmac.Text = (20.00).ToString();

//txtGravel.Text = (29.75).ToString();

//btnStart.Focus();

//

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

DialogResult answer = MessageBox.Show(

"Are you sure that you want to exit this application?",

"Exit", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (answer == DialogResult.Yes)

Close();

}

/// <summary>

/// validate exchang rate and prices of materials

/// change view to dayly work

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnStart\_Click(object sender, EventArgs e)

{

if (validateNumericTextBox(txtExchangeRate, out exchangeRate, "Exchange Rate", 3, .5m, 4) &&

validateNumericTextBox(txtBrick, out pricesOfMaterials[0], "Price of Brick") &&

validateNumericTextBox(txtConcrete, out pricesOfMaterials[1], "Price of Concrete") &&

validateNumericTextBox(txtTarmac, out pricesOfMaterials[2], "Price of Tarmac") &&

validateNumericTextBox(txtGravel, out pricesOfMaterials[3], "Price of Gravel"))

{

grpFoundations.Visible = true;

pnlCalculate.Visible = true;

pnlPrice.Visible = false;

pnlExchangeRate.Visible = false;

lblEuroDrivesOrPatios.ForeColor = Color.Green;

btnStart.Visible = false;

btnCalculate.Visible = true;

btnClear.Visible = true;

txtLength.Focus();

txtGBR.Text = "1";

txtEuro.Text = exchangeRate.ToString();

setCurrentPrice();

//lblOutput.Text = priceCurrentMaterial.ToString();

if (rbStandard.Checked)

foundations = 1;

else

foundations = 1.25m;

}

}

/// <summary>

/// set choisen material price

/// and material name

/// </summary>

private void setCurrentPrice()

{

if (rbBrick.Checked)

{

priceCurrentMaterial = pricesOfMaterials[0];

materialName = "Brick";

}

else if (rbConcrete.Checked)

{

priceCurrentMaterial = pricesOfMaterials[1];

materialName = "Contrete";

}

else if (rbTarmac.Checked)

{

priceCurrentMaterial = pricesOfMaterials[2];

materialName = "Tarmac";

}

else if (rbGravel.Checked)

{

priceCurrentMaterial = pricesOfMaterials[3];

materialName = "Gravel";

}

}

/// <summary>

/// validate text in TextBox

/// if is valid return true and value in out variable "value"

/// property, range and precision are optional

/// </summary>

/// <param name="textBox"></param>

/// <param name="value"></param>

/// <param name="property"></param>

/// <param name="topRange"></param>

/// <param name="bottomRange"></param>

/// <param name="precision"></param>

/// <returns>"true or false"</returns>

private Boolean validateNumericTextBox(TextBox textBox, out decimal value,

string property="Price", decimal topRange=1000, decimal bottomRange=1, int precision=2)

{

string text = textBox.Text.Trim();

// empty text box

if (text.Length == 0) {

value = 0;

reminderMessageBox(textBox, "Please enter the " + property);

return false;

}

// not numeric value

if (! decimal.TryParse(text, out value))

{

reminderMessageBox(textBox, "Please enter the numeric value");

return false;

}

// validate range of value

if (value > topRange || value < bottomRange)

{

value = 0;

reminderMessageBox(textBox, "Please enter the value bettwen " +

bottomRange + " and " + topRange);

return false;

}

// check precision

decimal correctPrecision = Math.Round(value, precision, MidpointRounding.AwayFromZero);

if (value != correctPrecision)

{

reminderMessageBox(textBox, "Incorect precision of " + property );

textBox.Text = correctPrecision.ToString();

textBox.SelectAll();

return false;

}

return true;

}

/// <summary>

/// MessageBox with OK button and exclamation icon

/// </summary>

/// <param name="textBox"></param>

/// <param name="message"></param>

private void reminderMessageBox(TextBox textBox, string message)

{

MessageBox.Show(message, "Reminder",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

textBox.Clear();

textBox.Focus();

}

// Focus Leave Events for all TextBox-es

private void txtExchangeRate\_Leave(object sender, EventArgs e)

{

validateNumericTextBox(txtExchangeRate, out exchangeRate, "Exchange Rate", 3, .5m, 4);

}

private void txtBrick\_Leave(object sender, EventArgs e)

{

validateNumericTextBox(txtBrick, out pricesOfMaterials[0]);

}

private void txtConcrete\_Leave(object sender, EventArgs e)

{

validateNumericTextBox(txtConcrete, out pricesOfMaterials[1]);

}

private void txtTarmac\_Leave(object sender, EventArgs e)

{

validateNumericTextBox(txtTarmac, out pricesOfMaterials[2]);

}

private void txtGravel\_Leave(object sender, EventArgs e)

{

validateNumericTextBox(txtGravel, out pricesOfMaterials[3]);

}

private void txtLength\_Leave(object sender, EventArgs e)

{

decimal length;

validateNumericTextBox(txtLength, out length, "Length");

}

private void txtWidth\_Leave(object sender, EventArgs e)

{

decimal width;

validateNumericTextBox(txtWidth, out width, "Width");

}

/// <summary>

/// Event for 4 radio battons on change

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbMaterial\_CheckedChanged(object sender, EventArgs e)

{

//int index;

//int.TryParse((((RadioButton)sender).Tag).ToString(), out index);

//priceCurrentMaterial = pricesOfMaterials[index];

//lblOutput.Text = priceCurrentMaterial.ToString();

setCurrentPrice();

}

/// <summary>

/// radio buttons for foudations

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void rbFoundations\_CheckedChanged(object sender, EventArgs e)

{

if (sender == rbStandard)

{

foundations = 1;

foundationName = "Standard";

}

else

{

foundations = 1.25m;

foundationName = "Extra Deep";

}

}

/// <summary>

/// calculate price for area

/// based on the daily values entered for materials

/// and exchange rates

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnCalculate\_Click(object sender, EventArgs e)

{

decimal area;

decimal priceGBP;

decimal priceEuro;

if (validateArea(out area))

{

lblOutput.BackColor = Color.White;

lblOutput.Text = String.Format("{0} selected with {1} foundation", materialName, foundationName);

priceGBP = area \* priceCurrentMaterial \* foundations;

priceEuro = Math.Round(priceGBP \* exchangeRate, 2, MidpointRounding.AwayFromZero);

priceGBP = Math.Round(priceGBP, 2, MidpointRounding.AwayFromZero);

txtGBR.Text = String.Format("£ {0:N2}", priceGBP);

txtEuro.Text = String.Format("€ {0:N2}", priceEuro);

btnClear.Focus();

}

}

/// <summary>

/// calculate area

/// if invalid field return false

/// </summary>

/// <param name="area"></param>

/// <returns></returns>

private bool validateArea(out decimal area)

{

decimal length;

decimal width;

if (validateNumericTextBox(txtLength, out length, "Length") &&

validateNumericTextBox(txtWidth, out width, "Width"))

{

area = length \* width;

return true;

}

area = 0;

return false;

}

/// <summary>

/// reset view

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnClear\_Click(object sender, EventArgs e)

{

lblOutput.BackColor = Color.Green;

lblOutput.Text = "";

txtGBR.Text = "1";

txtEuro.Text = exchangeRate.ToString();

txtWidth.Clear();

txtLength.Text = "";

rbBrick.Select();

rbStandard.Select();

txtLength.Focus();

}

}

}