Paul Garibay

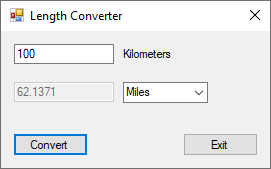
CSC 566

Project 2

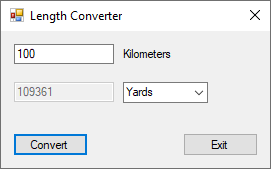
**Problem 1**



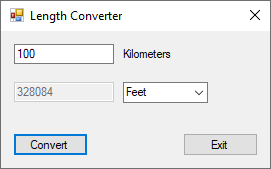
*Class Diagram*



*Screen 1 - Kilometers to Miles*



*Screen 2 - Kilometers to Yards*



*Screen 3 - Kilometers to Feet*

abstract class Length

{

public enum Units { None, Kilometers, Miles, Yards, Feet };

protected Length successor;

public void SetSuccessor(Length successor)

{

this.successor = successor;

}

public abstract string HandleRequest(Units unitsToConvertTo);

}

class Miles: Length

{

private string value;

public Miles(string value)

{

this.value = value;

}

public override string HandleRequest(Units unitsToConvertTo)

{

double numValue = 0;

if (double.TryParse(value, out numValue))

{

if (unitsToConvertTo == Units.Miles)

{

return (numValue \* 0.621371).ToString();

}

else if (successor != null)

{

return successor.HandleRequest(unitsToConvertTo);

}

}

return value;

}

}

class Yards : Length

{

private string value;

public Yards(string value)

{

this.value = value;

}

public override string HandleRequest(Units unitsToConvertTo)

{

double numValue = 0;

if (double.TryParse(value, out numValue))

{

if (unitsToConvertTo == Units.Yards)

{

return (numValue \* 1093.61).ToString();

}

else if (successor != null)

{

return successor.HandleRequest(unitsToConvertTo);

}

}

return value;

}

}

class Feet : Length

{

private string value;

public Feet(string value)

{

this.value = value;

}

public override string HandleRequest(Units unitsToConvertTo)

{

double numValue = 0;

if (double.TryParse(value, out numValue))

{

if (unitsToConvertTo == Units.Feet)

{

return (numValue \* 3280.84).ToString();

}

else if (successor != null)

{

return successor.HandleRequest(unitsToConvertTo);

}

}

return value;

}

}

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

dlUnits.SelectedIndex = 0;

}

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

{

Close();

}

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

{

// Assure a valid value

string value = tbKilometers.Text;

try

{

double.Parse(value);

}

catch

{

tbConvertedLength.Text = string.Empty;

MessageBox.Show("Please enter a valid length.", "Error", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

// Get the units to convert to

Length.Units newUnit;

switch (dlUnits.Text.ToLower())

{

case "miles": newUnit = Length.Units.Miles; break;

case "yards": newUnit = Length.Units.Yards; break;

case "feet": newUnit = Length.Units.Feet; break;

default: newUnit = Length.Units.None; break;

}

// Setup the Chain of Responsibility

Miles miles = new Miles(value);

Yards yards = new Yards(value);

Feet feet = new Feet(value);

miles.SetSuccessor(yards);

yards.SetSuccessor(feet);

// Convert kilometers to new units

value = miles.HandleRequest(newUnit);

// Display the converted value

tbConvertedLength.Text = value;

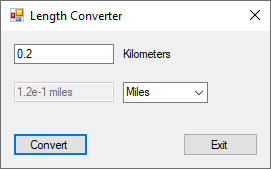
}

}

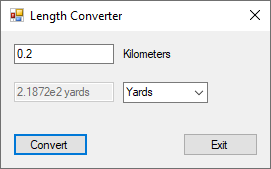
**Problem 2**



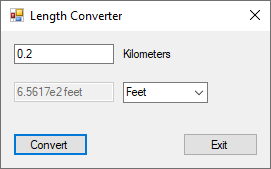
*Class Diagram*



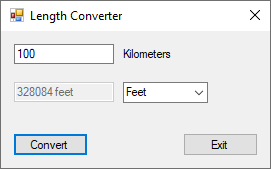
*Screen 1 - Kilometers to Miles*



*Screen 2 - Kilometers to Yards*



*Screen 3 - Kilometers to Feet*



*Screen 4 - Kilometers to Yards (no exponent format needed)*

**Note:** Length, Miles, Yards, and Feet classes are exactly the same as in problem 1. Therefore, the

code for these classes have not been included here.

abstract class FormatDecorator

{

protected string value;

public abstract string Format();

}

class RoundingDecorator : FormatDecorator

{

public RoundingDecorator(string value)

{

this.value = value;

}

public override string Format()

{

double numberValue = 0;

if (double.TryParse(value, out numberValue))

{

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

return numberValue.ToString();

}

return value;

}

}

class ExponentDecorator : FormatDecorator

{

public ExponentDecorator(string value)

{

this.value = value;

}

public override string Format()

{

// Find decimal position and remove it.

int decimalIndex = value.IndexOf('.');

if (decimalIndex > -1)

{

value = value.Remove(decimalIndex, 1);

}

if (value[0] != '0')

{

if (decimalIndex > 0)

{

// Calculate how many places to the left the decimal must move

int numberOfDecimalMoves = decimalIndex - 1;

if (numberOfDecimalMoves >= 0)

{

// Format the string into positive exponent notation

value = value.Insert(1, ".");

return value.Insert(value.Length, "e" +

numberOfDecimalMoves.ToString());

}

}

}

else

{

// Calculate how many places to the right the decimal must move

while (decimalIndex < value.Length)

{

if (value[decimalIndex] != '0')

{

break;

}

else

{

decimalIndex++;

}

}

// Remove leading zeros

value = value.Remove(0, decimalIndex);

// Format the string into negative exponent notation

value = value.Insert(1, ".");

return value.Insert(value.Length, "e-" + decimalIndex.ToString());

}

return value;

}

}

class UnitDecorator: FormatDecorator

{

Length.Units unitsToConvertTo;

public UnitDecorator(string value, Length.Units unitsToConvertTo)

{

this.value = value;

this.unitsToConvertTo = unitsToConvertTo;

}

public override string Format()

{

return value + " " + unitsToConvertTo.ToString();

}

}

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

dlUnits.SelectedIndex = 0;

}

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

{

Close();

}

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

{

// Assure a valid value

string value = tbKilometers.Text;

try

{

double.Parse(value);

}

catch

{

tbConvertedLength.Text = string.Empty;

MessageBox.Show("Please enter a valid length.", "Error", MessageBoxButtons.OK,

MessageBoxIcon.Exclamation);

return;

}

// Get the units to convert to

Length.Units newUnit;

switch (dlUnits.Text.ToLower())

{

case "miles": newUnit = Length.Units.Miles; break;

case "yards": newUnit = Length.Units.Yards; break;

case "feet": newUnit = Length.Units.Feet; break;

default: newUnit = Length.Units.None; break;

}

// Setup the Chain of Responsibility

Miles miles = new Miles(value);

Yards yards = new Yards(value);

Feet feet = new Feet(value);

miles.SetSuccessor(yards);

yards.SetSuccessor(feet);

// Convert kilometers to new units

value = miles.HandleRequest(newUnit);

// Decorate the converted value with the formats

RoundingDecorator roundingDecorator = new RoundingDecorator(value);

value = roundingDecorator.Format();

ExponentDecorator exponentDecorator = new ExponentDecorator(value);

value = exponentDecorator.Format();

UnitDecorator unitDecorator = new UnitDecorator(value, newUnit);

value = unitDecorator.Format();

// Display the converted value

tbConvertedLength.Text = value;

}

}