|  |  |
| --- | --- |
| Business Application Programming – MS806  **Assignment - 4** | **Assignment 5**  TeamBuilder v2  **Student and Submission Details**  **Student Name: Siddhant Kumar Kandoi Student Id: 19231361 Subject Code: MS806 Date: 01/12/2019 Assignment Number: 5 Total Pages: 10** |
|  |  |

//

/\* Student Name: Siddhant Kumar Kandoi

\* Student ID: 19231361

\* Course Name: Masters Information Systems Management

\* Date: 01/12/2019

\* Assignment Number: 5

\* Assignment: TeamBuilder v2

\* Naming Convention: PascalCasing; For each control I have tried to use the similar naming pattern

\* i.e. Title of the form + Functionality + Control Type + Numeric digit

\*/

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Kandoi\_SiddhantKumar\_Assignment5\_MS806

{

public partial class TeamBuilder : Form

{

public TeamBuilder()

{

InitializeComponent();

}

//

// Declaring global variables which is used at every click of Add to Booking, Complete Order and Report Generation buttons

int NumberOfPlaces;

int AvailablePlaces;

decimal TotalBookingCost;

decimal ItemCost=0m;

string BookingMessage = "";

string DaySaleSummary = "";

//

// Declaring Constant for Event - Location Pricing

readonly static decimal[,] EVENTPRICING = {{1100m, 930m, 1050m, 1210m, 790m},

{1750m, 1495m, 1675m, 1915m, 1285m},

{2500m, 2160m, 2400m, 2720m, 1880m},

{3300m, 2790m, 3150m, 3630m, 2370m},

{1099m, 929m, 1049m, 1209m, 789m},

{2249m, 1824m, 2124m, 2524m, 1474m},

{1369m, 1114m, 1294m, 1534m, 904m},

{999m, 829m, 949m, 1109m, 689m},

{3499m, 2989m, 3349m, 3829m, 2569m},

{2049m, 1624m, 1924m, 2324m, 1274m}};

//

// Declaring Constant for Meal - Location Pricing

readonly static decimal[,] MEALPRICING = { {99m, 75m, 24m, 0m},

{149m, 113m, 36m, 0m},

{198m, 150m, 48m, 0m},

{297m, 225m, 72m, 0m},

{99m, 75m, 24m, 0m},

{248m, 188m, 60m, 0m},

{149m, 113m, 36m, 0m},

{99m, 75m, 24m, 0m},

{297m, 225m, 72m, 0m},

{248m, 188m, 60m, 0m}};

//

// Declaring initialised value for Places in stock - Location matrix in case the file is not available then the application will use this initialized value

static int[,] PlacesAvailable = { {35, 67, 12, 77, 32},

{28, 3, 34, 12, 7},

{23, 2, 6, 4, 3},

{12, 6, 9, 4, 6},

{2, 0, 7, 5, 8},

{1, 8, 4, 7, 4},

{16, 24, 40, 4, 12},

{3, 7, 45, 3, 3},

{45, 12, 56, 12, 23},

{0, 0, 3, 0, 0} };

//

// To keep track of the changes before order is completed and may be required if user doesn't confirm the order

readonly static int[,] PlacesAvailableBeforeConfirmation = { {35, 67, 12, 77, 32},

{28, 3, 34, 12, 7},

{23, 2, 6, 4, 3},

{12, 6, 9, 4, 6},

{2, 0, 7, 5, 8},

{1, 8, 4, 7, 4},

{16, 24, 40, 4, 12},

{3, 7, 45, 3, 3},

{45, 12, 56, 12, 23},

{0, 0, 3, 0, 0} };

//

// Declaring constant for the Events Name

static string[] EVENTSNAME = { "Murder Mystery Weekend", "CSI Weekend", "The Great Outdoors", "The Chase", "Digital Refresh",

"Action Photography", "Team Ryder Cup", "Abseiling", "War Games", "Find Wally" };

//

// Declaring constant for the Meal Types

static string[] MEALTYPES = { "Full Meal", "Half Meal", "Breakfast", "No Meal" };

//

// Declaring constant for the Location Names

static string[] LOCATIONNAMES = { "Cork", "Dublin", "Galway", "Belmullet", "Belfast" };

//

// Declaring constant for the Places inventory file name

const string PLACES\_FILENAME = "PlacesInventory.txt";

//

// Declaring constant for the directory to check for transaction number

const string TransactionFileDirectory = @"C:\Users\kando\Desktop\Ireland\Kandoi\_SiddhantKumar\_Assignment5\_MS806\Kandoi\_SiddhantKumar\_Assignment5\_MS806\bin\Debug";

//

// Event and Action defined for the load of Application Page

private void TeamBuilder\_Load(object sender, EventArgs e)

{

//

// Code to search for inventory file and load the inventory data into Place inventory arrays

if (File.Exists(PLACES\_FILENAME))

{

String input = File.ReadAllText(PLACES\_FILENAME);

int i = 0, j = 0;

foreach (var row in input.Split('\n'))

{

if (i == 10)

{

break;

}

j = 0;

foreach (var col in row.Trim().Split('\t'))

{

PlacesAvailable[i, j] = int.Parse(col.Trim());

PlacesAvailableBeforeConfirmation[i, j] = int.Parse(col.Trim());

j++;

}

i++;

}

}

//

// Button disabled

CompleteOrderButton1.Enabled = false;

AddToCartButton1.Enabled = false;

}

//

// Method for calculating the price based on selection

private void ItemPricingMethod()

{

//

// To check if event and location are selected

if (EventListBox1.SelectedIndex != -1 && LocationListBox1.SelectedIndex != -1)

{

//

// To check if Number of Places is entered

if (NumberOfPlacesTextBox1.Text != "")

{

try

{

NumberOfPlaces = int.Parse(NumberOfPlacesTextBox1.Text);

//

// Handling negative values in quantity

if (NumberOfPlaces <= 0)

{

MessageBox.Show("Investment amount should be positive value, setting the default value as 1", "I/P Error", MessageBoxButtons.RetryCancel, MessageBoxIcon.Error);

//

// Bringing the focus back to number of places textbox for user edit

NumberOfPlacesTextBox1.Text = "1";

NumberOfPlaces = 1;

NumberOfPlacesTextBox1.Focus();

NumberOfPlacesTextBox1.SelectAll();

}

else

{

if (MealOptionsListBox1.SelectedIndex != -1)

{

ItemCost = (EVENTPRICING[EventListBox1.SelectedIndex, LocationListBox1.SelectedIndex] + MEALPRICING[EventListBox1.SelectedIndex, MealOptionsListBox1.SelectedIndex]) \* NumberOfPlaces;

}

else

{

MealOptionsListBox1.SelectedIndex = 3;

ItemCost = (EVENTPRICING[EventListBox1.SelectedIndex, LocationListBox1.SelectedIndex]) \* NumberOfPlaces;

}

AddToCartButton1.Enabled = true;

TotalItemCostValueLabel1.Text = ItemCost.ToString() + String.Format("\u20AC");

}

}

catch

{

NumberOfPlacesTextBox1.Text = "1";

NumberOfPlaces = 1;

MessageBox.Show("Number of places should be valid positive integer value, setting the default places to 1", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);

NumberOfPlacesTextBox1.Focus();

NumberOfPlacesTextBox1.SelectAll();

}

}

else

{

NumberOfPlacesTextBox1.Text = "1";

NumberOfPlaces = 1;

if (MealOptionsListBox1.SelectedIndex != -1)

{

ItemCost = (EVENTPRICING[EventListBox1.SelectedIndex, LocationListBox1.SelectedIndex] + MEALPRICING[EventListBox1.SelectedIndex, MealOptionsListBox1.SelectedIndex]);

}

else

{

MealOptionsListBox1.SelectedIndex = 3;

ItemCost = (EVENTPRICING[EventListBox1.SelectedIndex, LocationListBox1.SelectedIndex]);

}

AddToCartButton1.Enabled = true;

TotalItemCostValueLabel1.Text = ItemCost.ToString() + String.Format("\u20AC");

}

}

}

//

// Method to calculate the available places based on event and location selection

private void AvailablePlacesEventLocationMethod()

{

if (EventListBox1.SelectedIndex != -1 && LocationListBox1.SelectedIndex != -1)

{

LocationAvailableListbox1.Items.Clear();

LocationAvailableListbox1.Items.Add(EVENTSNAME[EventListBox1.SelectedIndex] + "\t\t" + LOCATIONNAMES[LocationListBox1.SelectedIndex] + "\t\t" + PlacesAvailable[EventListBox1.SelectedIndex, LocationListBox1.SelectedIndex]);

}

}

//

// event and action when location list box index changes

private void LocationListBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

ItemPricingMethod();

AvailablePlacesEventLocationMethod();

}

//

// event and action when Meal list box index changes

private void MealOptionsListBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

ItemPricingMethod();

}

//

// event and action when number of places text box changes

private void NumberOfPlacesTextBox1\_TextChanged(object sender, EventArgs e)

{

ItemPricingMethod();

}

//

// event and action when event list box index changes

private void EventListBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

ItemPricingMethod();

AvailablePlacesEventLocationMethod();

}

//

// Method defiened for checking the uniqueness of each random number which is getting generated. This method is being used by another method GenerateUniqueTransnum

private Boolean CheckUnique(string Transnum, string FileDirectory)

{

Boolean Unique = false;

StreamReader InputFile;

string[] filePaths = Directory.GetFiles(FileDirectory);

foreach (string filePath in filePaths)

{

InputFile = File.OpenText(filePath);

while (!InputFile.EndOfStream)

{

if (Transnum.Equals(InputFile.ReadLine()))

{

InputFile.Close();

return Unique;

}

}

InputFile.Close();

}

Unique = true;

return Unique;

}

//

// Method defiened for generating the random 6 digit number and also calling the Unique method defined above to check the uniqueness

private string GenerateUniqueTransnum()

{

Random generator = new Random();

string Transnum;

Boolean IsThisUnique;

do

{

Transnum = generator.Next(0, 999999).ToString("D6");

IsThisUnique = CheckUnique(Transnum, TransactionFileDirectory);

} while (!IsThisUnique);

return Transnum;

}

//

// event and action defiened when user clicks on Complete order button

private void CompleteOrderButton1\_Click(object sender, EventArgs e)

{

string CompletedBookingMessage;

string TransactionNumber = GenerateUniqueTransnum();

// Message to show to user

CompletedBookingMessage = "Transaction No.:" + TransactionNumber + "\n" + BookingMessage + "Total Booking Cost:" + TotalBookingCost + String.Format("\u20AC") + "\n\n";

//

// Copying the changes of places available to confirmed places available for confirmation

if ((MessageBox.Show(CompletedBookingMessage, "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Information) == DialogResult.Yes))

{

Array.Copy(PlacesAvailable, PlacesAvailableBeforeConfirmation, PlacesAvailable.Length);

DaySaleSummary = DaySaleSummary + CompletedBookingMessage;

ClearButton1\_Click(sender, e);

}

}

//

// event and action defiened for the click of add to cart button

private void AddToCartButton1\_Click(object sender, EventArgs e)

{

AddToCartButton1.Enabled = false;

AvailablePlaces = PlacesAvailable[EventListBox1.SelectedIndex, LocationListBox1.SelectedIndex];

//

// to check if available places is greater than 0

if (AvailablePlaces != 0)

{

//

// To check if number of places entered is less than available

if ((AvailablePlaces - NumberOfPlaces) >= 0)

{

PlacesAvailable[EventListBox1.SelectedIndex, LocationListBox1.SelectedIndex] = AvailablePlaces - NumberOfPlaces;

TotalBookingCost = TotalBookingCost + ItemCost;

TotalBookingAmountAnsLabel1.Text = TotalBookingCost + String.Format("\u20AC");

BookingListBox1.Items.Add(EVENTSNAME[EventListBox1.SelectedIndex] + "\t" + LOCATIONNAMES[LocationListBox1.SelectedIndex] + "\t" + NumberOfPlaces + "\t" + ItemCost + String.Format("\u20AC"));

CompleteOrderButton1.Enabled = true;

BookingMessage = BookingMessage + "Event Details:\n1.Event Name:" + EVENTSNAME[EventListBox1.SelectedIndex] + "\n2.Location:" + LOCATIONNAMES[LocationListBox1.SelectedIndex] + "\n3.Meal Option Selected:" + MEALTYPES[MealOptionsListBox1.SelectedIndex] + "\n4.Total Number of Places Selected:" + NumberOfPlaces + "\n5.Total Event Cost:" + ItemCost + "\n\n";

EventListBox1.ClearSelected();

LocationListBox1.ClearSelected();

MealOptionsListBox1.ClearSelected();

NumberOfPlacesTextBox1.Text = "";

TotalItemCostValueLabel1.Text = "";

LocationAvailableListbox1.Items.Clear();

}

//

// to set the number of places to the maximum available stock

else

{

NumberOfPlacesTextBox1.Text = AvailablePlaces.ToString();

MessageBox.Show("Number of places you entered exceed the available stock. There are only " + AvailablePlaces + " available", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);

NumberOfPlacesTextBox1.Focus();

NumberOfPlacesTextBox1.SelectAll();

}

}

else

{

MessageBox.Show("Unfortunately, There are no available place for the event and location. Please revise the booking", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

//

// Event and action defiened for the click of clear button

private void ClearButton1\_Click(object sender, EventArgs e)

{

EventListBox1.ClearSelected();

LocationListBox1.ClearSelected();

AddToCartButton1.Enabled = false;

CompleteOrderButton1.Enabled = false;

NumberOfPlacesTextBox1.Text = "";

TotalItemCostValueLabel1.Text = "";

MealOptionsListBox1.ClearSelected();

LocationAvailableListbox1.Items.Clear();

//

// Code to get back the number of places which got changes due to add in cart option but user has not confirmed the booking

if(BookingListBox1.Items.Count > 0)

{

Array.Copy(PlacesAvailableBeforeConfirmation, PlacesAvailable, PlacesAvailableBeforeConfirmation.Length);

BookingListBox1.Items.Clear();

}

NumberOfPlacesTextBox1.Text = "1";

TotalBookingCost = 0m;

ItemCost = 0m;

BookingMessage = "";

TotalBookingAmountAnsLabel1.Text = "";

}

//

// Method defiened when user tries to close the application and application saves the transaction details for the day and as wll as places available after sales

private void DayClosing()

{

string DaySummaryFile;

try

{

StreamWriter PlacesAvailableDayEndTextWrite, textWrite;

PlacesAvailableDayEndTextWrite = File.CreateText(PLACES\_FILENAME);

for (int i = 0; i < EVENTSNAME.Length; i++)

{

for (int j = 0; j < LOCATIONNAMES.Length; j++)

{

PlacesAvailableDayEndTextWrite.Write(PlacesAvailableBeforeConfirmation[i, j] + "\t");

}

PlacesAvailableDayEndTextWrite.Write("\n");

}

PlacesAvailableDayEndTextWrite.Flush();

PlacesAvailableDayEndTextWrite.Close();

//

// to save the data into transaction file for the day

DaySummaryFile = "EOD\_" + DateTime.Now.ToString("dd.MM.yyy") + "\_BOOKING\_REPORT.txt";

//

// To check if file exists or not

if (File.Exists(DaySummaryFile))

{

textWrite = File.AppendText(DaySummaryFile);

foreach (string str in Regex.Split(DaySaleSummary, "\n"))

textWrite.WriteLine(str);

}

else

{

textWrite = File.CreateText(DaySummaryFile);

foreach (string str in Regex.Split(DaySaleSummary, "\n"))

textWrite.WriteLine(str);

}

textWrite.Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

//

// Event and action for the click of exit button

private void ExitButton1\_Click(object sender, EventArgs e)

{

//

// If condition to confirm with user if the user really wants to exit or not, if yes the application is closed

if ((MessageBox.Show("Are you sure you want to exit?", "Exit Warning", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) == DialogResult.Yes))

{

Application.Exit();

}

}

//

// Event and action for the application when user is closing the application

private void TeamBuilder\_FormClosing(object sender, FormClosingEventArgs e)

{

MessageBox.Show("Your sale summary for today is being saved", "Exit Information", MessageBoxButtons.OK, MessageBoxIcon.Warning);

if (DaySaleSummary != "")

{

DayClosing();

}

}

//

// Event when anyone clicks on Number of places text box

private void NumberOfPlacesTextBox1\_MouseClick(object sender, MouseEventArgs e)

{

NumberOfPlacesTextBox1.SelectAll();

}

//

// Method coded for generating the detailed summary

private void DetailedSummaryReport()

{

string SummaryReportFileName = DateTime.Now.ToString("dd.MM.yyy") + "\_DETAILED\_SUMMARY\_REPORT.txt";

try

{

StreamWriter TextWrite;

//

// to check if any transaction happened for that session, if yes than transactional data would also be included in the report

if (DaySaleSummary != "")

{

if (File.Exists(SummaryReportFileName))

{

TextWrite = File.AppendText(SummaryReportFileName);

foreach (string str in Regex.Split(DaySaleSummary, "\n"))

TextWrite.WriteLine(str);

TextWrite.WriteLine("Places In Stock:");

for (int i = 0; i < EVENTSNAME.Length; i++)

{

for (int j = 0; j < LOCATIONNAMES.Length; j++)

{

TextWrite.Write(PlacesAvailableBeforeConfirmation[i, j] + "\t");

}

TextWrite.Write("\n");

}

TextWrite.Flush();

TextWrite.Close();

}

else

{

TextWrite = File.CreateText(SummaryReportFileName);

foreach (string str in Regex.Split(DaySaleSummary, "\n"))

TextWrite.WriteLine(str);

TextWrite.WriteLine("Places In Stock:");

for (int i = 0; i < EVENTSNAME.Length; i++)

{

for (int j = 0; j < LOCATIONNAMES.Length; j++)

{

TextWrite.Write(PlacesAvailableBeforeConfirmation[i, j] + "\t");

}

TextWrite.Write("\n");

}

}

TextWrite.Close();

MessageBox.Show("Report Generated, Details of completed booking in this session and places available have been saved in the" + SummaryReportFileName, "INFORMATION", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

if (File.Exists(SummaryReportFileName))

{

TextWrite = File.AppendText(SummaryReportFileName);

TextWrite.WriteLine("Places In Stock:");

for (int i = 0; i < EVENTSNAME.Length; i++)

{

for (int j = 0; j < LOCATIONNAMES.Length; j++)

{

TextWrite.Write(PlacesAvailableBeforeConfirmation[i, j] + "\t");

}

TextWrite.Write("\n");

}

}

else

{

TextWrite = File.CreateText(SummaryReportFileName);

TextWrite.WriteLine("Places In Stock:");

for (int i = 0; i < EVENTSNAME.Length; i++)

{

for (int j = 0; j < LOCATIONNAMES.Length; j++)

{

TextWrite.Write(PlacesAvailableBeforeConfirmation[i, j] + "\t");

}

TextWrite.Write("\n");

}

}

TextWrite.Close();

MessageBox.Show("Report Generated, there was no cpmpleted booking in this session. Details of places available has been saved in the" + SummaryReportFileName, "INFORMATION", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

//

// Event defiened when someone clicks on GenerateReportButton

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

{

DetailedSummaryReport();

}

}

}