**Course Project – Error Handling**

Tim Mastarone

Rasmussen University

Microsoft C# Programming

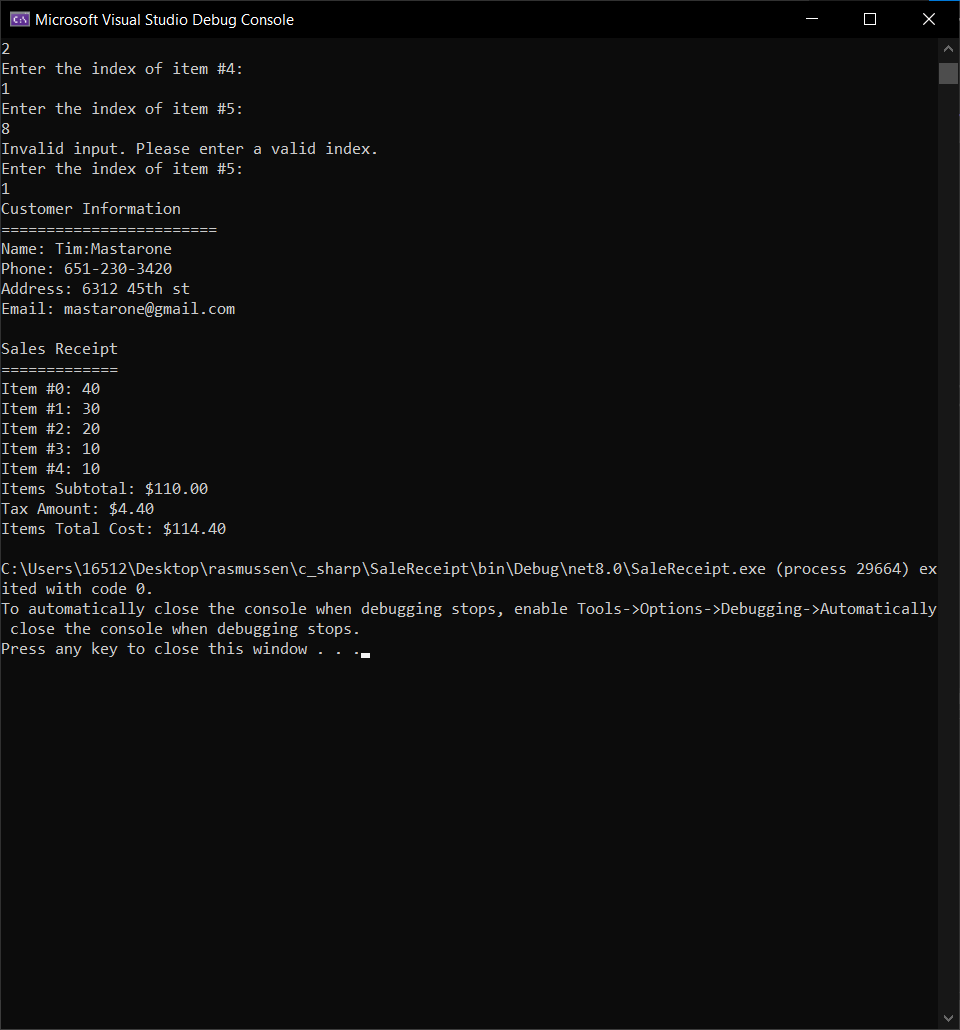
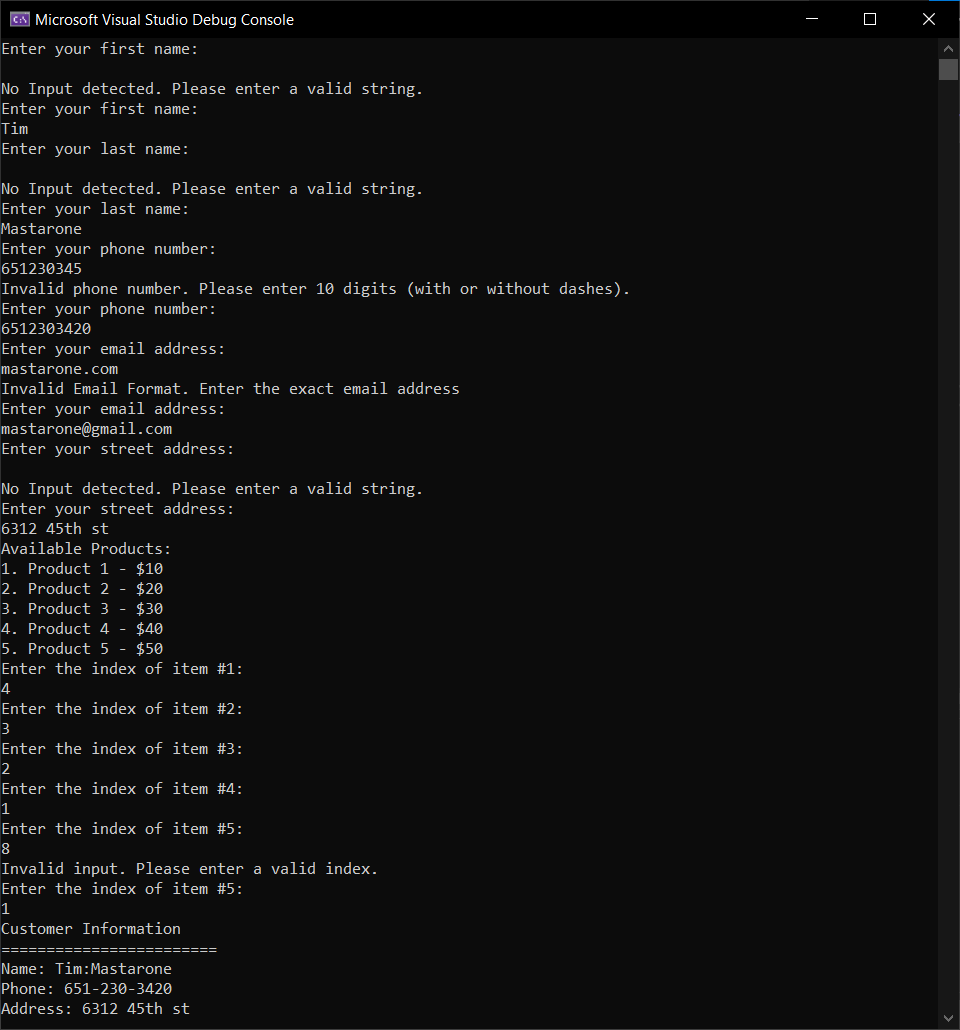
Instructor: Jim Barringer

Course Project – Module 03

April 18, 2024

I refactored the SalesReceipt program to have increased error handling. While I was taking care of some user input validation, I now have a function for validating each ‘type’ of data. For plain string such as the names, and addresses I am checking that the string is not null or blank using shared function. For the phone numbers and email addresses I have added regular expression to check each fits a set format. Each of these validation checks are inside of a loop that continues until the input is validated then it is saved to the customer object.

In the screenshot below, I entered an invalid value for each prompt followed by an acceptable input:



The code below is my SalesReceipt.cs file:

using System.Text.RegularExpressions;

using static System.Runtime.InteropServices.JavaScript.JSType;

class Program

{

//4% fixed tax rate

const double taxRate = .04;

static double[] items = new double[5];

static string[] productNames = { "Product 1", "Product 2", "Product 3", "Product 4", "Product 5" };

static double[] productPrices = { 10.0, 20.0, 30.0, 40.0, 50.0 };

static void Main(string[] args)

{

Customer customer = new();

customer.FirstName = ValidateEnteredString("Enter your first name: ");//any string is acceptable, this function just validates something was entered

customer.LastName = ValidateEnteredString("Enter your last name: ");

customer.Phone = GetAndValidateNumber();//loops and makes sure phone number is 10 digits before continuing

customer.Email = GetAndValidateEmail();//loops to check email address format before continuing

customer.StAddress = ValidateEnteredString("Enter your street address: ");

// Display available products

Console.WriteLine("Available Products:");

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

{

Console.WriteLine($"{i + 1}. {productNames[i]} - ${productPrices[i]}");

}

// Loop 5 times for the items the customer purchased

for (int i = 0; i < 5; i++)

{

Console.WriteLine($"Enter the index of item #{i + 1}:");

string? userInput = Console.ReadLine();

if (int.TryParse(userInput, out int itemIndex) && itemIndex >= 1 && itemIndex <= productNames.Length)//validates input can be parsed and the index is in range

{

//index is valid, so we add the indexed value to the total

customer.ItemsTotal += productPrices[itemIndex - 1];

items[i] = productPrices[itemIndex - 1];

}

else

{

Console.WriteLine("Invalid input. Please enter a valid index.");

i--; // Decrement i to repeat the current iteration

}

}

OutputCustomerInfo(customer);

double taxAmount = CalculateTax(customer.ItemsTotal);

OutputSalesReceipt(customer, taxAmount);//calculates the tax, sub-total, and total and displays them

}

static double CalculateTax(double subtotal)

{

double tax;

tax = taxRate \* subtotal;

return tax;

}

private static string GetAndValidateNumber()

{

string? phoneNumber;

string phoneNumberNoDashes;

string phoneNumberRegex = @"^\d{10}$";

bool isValidPhoneNumber;

while (true)

{

Console.WriteLine("Enter your phone number: ");

phoneNumber = Console.ReadLine();

if (phoneNumber != null){

phoneNumberNoDashes = Regex.Replace(phoneNumber, @"[- ]", "");

isValidPhoneNumber = Regex.IsMatch(phoneNumber, phoneNumberRegex);

if (isValidPhoneNumber)

{

string formattedPhoneNumber = Regex.Replace(phoneNumberNoDashes, @"(\d{3})(\d{3})(\d{4})", "$1-$2-$3");

return formattedPhoneNumber;

}

else

{

Console.WriteLine("Invalid phone number. Please enter 10 digits (with or without dashes).");

}

}

else

{

Console.WriteLine("No Phone Number Entered. Please entered 10 digits (with or without dashes).");

}

}

}

private static string ValidateEnteredString(string message)

{

string? userInput;

while(true)

{

Console.WriteLine($"{message}");

userInput = Console.ReadLine();

if (userInput != null && userInput != "")

{

return userInput;

}

else

{

Console.WriteLine("No Input detected. Please enter a valid string.");

}

}

}

private static string GetAndValidateEmail()

{

string? emailEntered;

string emailRegex = @"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$";

bool isValidEmail;

while (true)

{

Console.WriteLine("Enter your email address: ");

emailEntered = Console.ReadLine();

if(emailEntered != null)

{

isValidEmail = Regex.IsMatch(emailEntered, emailRegex);

if (isValidEmail)

{

return emailEntered;

}

else

{

Console.WriteLine("Invalid Email Format. Enter the exact email address");

}

}

else

{

Console.WriteLine("No Email Entered. Please enter a valid email address.");

}

}

}

private static void OutputCustomerInfo(Customer customer)

{

Console.WriteLine("Customer Information");

Console.WriteLine("========================");

Console.WriteLine("Name: " + customer.FirstName + ":" + customer.LastName);

Console.WriteLine("Phone: " + customer.Phone);

Console.WriteLine("Address: " + customer.StAddress);

Console.WriteLine("Email: " + customer.Email);

Console.WriteLine("");

}

private static void OutputSalesReceipt(Customer customer, double taxTotal)

{

Console.WriteLine("Sales Receipt");

Console.WriteLine("=============");

for (int i = 0; i < 5; i++)

{

Console.WriteLine("Item #" + i.ToString() + ": " + items[i]);

}

Console.WriteLine("Items Subtotal: $" + customer.ItemsTotal.ToString("F2"));

Console.WriteLine("Tax Amount: $" + taxTotal.ToString("F2"));

Console.WriteLine("Items Total Cost: $" + (customer.ItemsTotal + taxTotal).ToString("F2"));

}

}

public class Customer

{

public string? FirstName { get; set; }

public string? LastName { get; set; }

public string? Phone { get; set; }

public string? StAddress { get; set; }

public string? Email { get; set; }

public double ItemsTotal { get; set; }

}