1

Corto 1

Marcelo Jose Contreras turcios, 201123097, E*scuela de Mecánica Eléctrica, Facultad de Ingeniería, https://github.com/marcelojct/Corto-1 Universidad de San Carlos de Guatemala*

***Resumen*—** Este proyecto consiste en desarrollar una aplicación de consola en Visual Basic .NET para calcular el costo del servicio de estacionamiento. La aplicación solicita al usuario datos como el nombre del cliente, NIT, identificación del vehículo, hora de entrada y salida. Con estos datos, calcula el monto a pagar aplicando una tarifa de Q15.00 para la primera hora y Q20.00 para las horas adicionales o fracciones. El programa valida las entradas del usuario para evitar errores y asegura que las transacciones se registren correctamente en un archivo de texto llamado facturas.txt. Este archivo almacena un resumen de la transacción, incluyendo detalles como el tiempo total en el estacionamiento y el monto a pagar. El proyecto incorpora principios clave de programación, como la manipulación de archivos, manejo de excepciones y validación de datos, proporcionando una solución práctica y eficiente para la gestión de estacionamientos.

I. Marco Teórico

1. Lenguaje de Programación: Visual Basic .NET
1.1. Historia y Evolución Visual Basic .NET (VB.NET) es un lenguaje de programación desarrollado por Microsoft que forma parte de la plataforma .NET. Fue lanzado en 2002 como una evolución del lenguaje Visual Basic clásico, con el objetivo de integrarse completamente en el entorno .NET Framework, lo que permitió a los desarrolladores crear aplicaciones de escritorio, web, y móviles con una mayor interoperabilidad y seguridad.
1.2. Características Principales

- **Sintaxis Simple y Clara**: Visual Basic .NET es conocido por su sintaxis simple y fácil de aprender, lo que lo convierte en una opción popular para principiantes.
- **Orientado a Objetos**: VB.NET es un lenguaje completamente orientado a objetos, lo que permite a los desarrolladores crear aplicaciones modulares y reutilizables.
- **Interoperabilidad**: Gracias a .NET Framework, VB.NET puede interoperar fácilmente con otros lenguajes de la plataforma .NET, como C# y F#.
- **Rápido Desarrollo de Aplicaciones (RAD)**: VB.NET es ideal para el desarrollo rápido de aplicaciones debido a su entorno de desarrollo integrado (IDE) Visual Studio, que ofrece herramientas como diseñadores de formularios, depuración integrada y asistentes de código.

2. Conceptos de Programación en Consola
2.1. Aplicaciones de Consola Una aplicación de consola es un tipo de programa que se ejecuta en la línea de comandos y no tiene una interfaz gráfica de usuario (GUI). Estas aplicaciones son ideales para tareas que requieren entrada y salida de datos de forma sencilla, como calculadoras, herramientas de línea de comandos, y scripts de automatización.
2.2. Interacción con el Usuario En las aplicaciones de consola, la interacción con el usuario se realiza a través de la entrada de texto (por medio del teclado) y la salida de texto (en la pantalla). Los comandos Console.ReadLine y Console.WriteLine en VB.NET permiten manejar la entrada y salida de datos respectivamente.
3. Manejo de Archivos en Visual Basic .NET
3.1. Manipulación de Archivos de Texto La manipulación de archivos de texto en VB.NET se realiza principalmente a través de las clases StreamReader y StreamWriter del espacio de nombres System.IO. Estas clases permiten leer y escribir archivos de texto de manera eficiente y segura, manejando también excepciones para evitar errores durante la lectura/escritura.
3.2. Persistencia de Datos La persistencia se refiere a la capacidad de un programa de almacenar datos que persistan más allá de la ejecución del programa. En este proyecto, la persistencia se logra mediante el almacenamiento de las facturas generadas en un archivo de texto (facturas.txt), que permite que los registros de transacciones sean accesibles para auditorías futuras.
4. Validación de Datos y Manejo de Errores
4.1. Importancia de la Validación de Datos La validación de datos es crucial para garantizar que la entrada del usuario sea correcta y significativa. En aplicaciones de consola, esto significa asegurarse de que los datos ingresados, como horas, números, y textos, sean del tipo y formato esperados para evitar errores en el procesamiento.
4.2. Manejo de Excepciones El manejo de excepciones en VB.NET permite capturar y manejar errores que pueden ocurrir durante la ejecución del programa, como intentar leer un archivo inexistente o un formato de hora incorrecto. Usando bloques Try...Catch, el programa puede responder a estos errores de manera controlada, proporcionando retroalimentación al usuario y evitando que la aplicación se cierre inesperadamente.
5. Cálculo de Tarifas y Redondeo
5.1. Redondeo de Horas En el contexto de un servicio de estacionamiento, el redondeo de horas implica convertir el tiempo fraccionado en una unidad entera, ya que se cobra por horas completas o fracciones de una hora como una hora completa. En VB.NET, se puede utilizar la función Math.Ceiling para redondear al alza cualquier valor decimal a su siguiente número entero.
5.2. Estructura de Tarifas Las tarifas en un sistema de estacionamiento pueden variar según el tiempo de uso. En este proyecto, se define una estructura simple de tarifas: una tarifa fija para la primera hora y una tarifa diferente para las horas adicionales. Este modelo de precios escalonados es común en servicios de estacionamiento y se puede implementar fácilmente mediante condicionales en el código.
6. Generación de Facturas
6.1. Contenido de la Factura Una factura debe incluir todos los detalles relevantes de la transacción, como el nombre del cliente, la identificación del vehículo, el tiempo total de estacionamiento, y el monto total a pagar. La factura sirve como un comprobante para el cliente y como un registro para el negocio.
6.2. Formato y Almacenamiento El formato de la factura en un archivo de texto debe ser claro y organizado, facilitando su comprensión y auditoría. Cada factura debe agregarse al archivo sin sobrescribir las facturas anteriores, lo que asegura que todas las transacciones sean registradas y puedan ser revisadas en el futuro.

II. Pistas Útiles

A. Objetivos

Objetivo General:
Desarrollar una aplicación de consola en Visual Basic .NET que permita calcular de manera eficiente y precisa el monto a pagar por el servicio de estacionamiento, generando y almacenando facturas detalladas de las transacciones en un archivo de texto.
Objetivos Específicos:

1. Implementar un sistema de cálculo de tarifas que contemple una tarifa fija para la primera hora de estacionamiento y una tarifa adicional por cada hora o fracción posterior, garantizando la precisión en el cálculo del monto total a pagar.
2. Desarrollar funciones de validación de datos para asegurar que las entradas del usuario, como las horas de entrada y salida, sean correctas y en el formato adecuado, reduciendo la posibilidad de errores durante la ejecución del programa.
3. Diseñar un módulo de generación de facturas que registre de forma clara y organizada los detalles de cada transacción en un archivo de texto, permitiendo un almacenamiento seguro y una fácil consulta futura de las facturas emitidas.

B. Marco Práctico

1. Validación de Datos y Manejo de Errores

1.1. Importancia de la Validación de Datos La validación de datos es crucial para garantizar que la entrada del usuario sea correcta y significativa. En aplicaciones de consola, esto significa asegurarse de que los datos ingresados, como horas, números, y textos, sean del tipo y formato esperados para evitar errores en el procesamiento.

1.2. Manejo de Excepciones El manejo de excepciones en VB.NET permite capturar y manejar errores que pueden ocurrir durante la ejecución del programa, como intentar leer un archivo inexistente o un formato de hora incorrecto. Usando bloques Try...Catch, el programa puede responder a estos errores de manera controlada, proporcionando retroalimentación al usuario y evitando que la aplicación se cierre inesperadamente.

2. Cálculo de Tarifas y Redondeo

2.1. Redondeo de Horas En el contexto de un servicio de estacionamiento, el redondeo de horas implica convertir el tiempo fraccionado en una unidad entera, ya que se cobra por horas completas o fracciones de una hora como una hora completa. En VB.NET, se puede utilizar la función Math.Ceiling para redondear al alza cualquier valor decimal a su siguiente número entero.

2.2. Estructura de Tarifas Las tarifas en un sistema de estacionamiento pueden variar según el tiempo de uso. En este proyecto, se define una estructura simple de tarifas: una tarifa fija para la primera hora y una tarifa diferente para las horas adicionales. Este modelo de precios escalonados es común en servicios de estacionamiento y se puede implementar fácilmente mediante condicionales en el código.

3 Generación de Facturas

3.1. Contenido de la Factura Una factura debe incluir todos los detalles relevantes de la transacción, como el nombre del cliente, la identificación del vehículo, el tiempo total de estacionamiento, y el monto total a pagar. La factura sirve como un comprobante para el cliente y como un registro para el negocio.

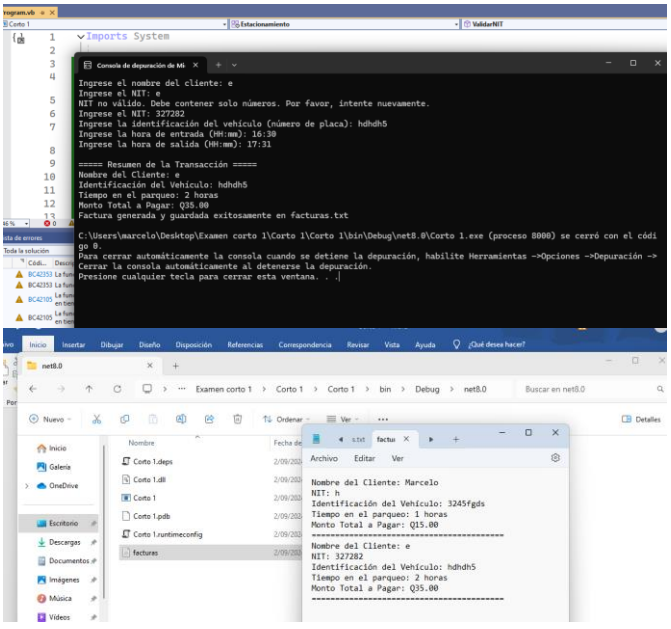
3.2. Formato y Almacenamiento El formato de la factura en un archivo de texto debe ser claro y organizado, facilitando su comprensión y auditoría. Cada factura debe agregarse al archivo sin sobrescribir las facturas anteriores, lo que asegura que todas las transacciones sean registradas y puedan ser revisadas en el futuro.

C. Código

```
Imports System.IO

0 referencias
Module Estacionamiento

    ' Función para calcular el monto total a pagar
    1 referencia
    Function CalcularMonto(horas As Integer) As Decimal
        If horas <= 1 Then
            Return 15
        Else
            Return 15 + (horas - 1) * 20
        End If
    End Function
```

```vb
' Función para validar entrada de hora
Function ValidarHora(mensaje As String) As DateTime
    Dim horaValida As DateTime
    While True
        Console.Write(mensaje)
        Dim entrada As String = Console.ReadLine()
        If DateTime.TryParse(entrada, horaValida) Then
            Return horaValida
        Else
            Console.WriteLine("Hora no válida. Por favor, intente nuevamente.")
        End If
    End While
End Function

' Función para validar que el NIT solo contenga números
Function ValidarNIT(mensaje As String) As String
    While True
        Console.Write(mensaje)
        Dim nit As String = Console.ReadLine()
        If IsNumeric(nit) Then
            Return nit
        Else
            Console.WriteLine("NIT no válido. Debe contener solo números. Por favor, intente nueva
        End If
    End While
End Function

' Función para escribir en el archivo facturas.txt
Sub EscribirFactura(nombre As String, nit As String, placa As String, horas As Integer, monto As D
    Dim rutaArchivo As String = "facturas.txt"
    Try
        Using escritor As StreamWriter = New StreamWriter(rutaArchivo, True)
            escritor.WriteLine("Nombre del Cliente: " & nombre)
            escritor.WriteLine("NIT: " & nit)
            escritor.WriteLine("Identificación del Vehículo: " & placa)
            escritor.WriteLine("Tiempo en el parqueo: " & horas & " horas")
            escritor.WriteLine("Monto Total a Pagar: Q" & monto.ToString("F2"))
            escritor.WriteLine("========================================")
        End Using
        Console.WriteLine("Factura generada y guardada exitosamente en " & rutaArchivo)
    Catch ex As Exception
        Console.WriteLine("Error al escribir en el archivo: " & ex.Message)
    End Try
End Sub

' Función principal
Sub Main()
    ' Ingreso de datos del cliente
    Console.Write("Ingrese el nombre del cliente: ")
    Dim nombre As String = Console.ReadLine()

    ' Validar que el NIT solo contenga números
    Dim nit As String = ValidarNIT("Ingrese el NIT: ")

    Console.Write("Ingrese la identificación del vehículo (número de placa): ")
    Dim placa As String = Console.ReadLine()

    ' Validación de hora de entrada y salida
    Dim horaEntrada As DateTime = ValidarHora("Ingrese la hora de entrada (HH:mm): ")
    Dim horaSalida As DateTime = ValidarHora("Ingrese la hora de salida (HH:mm): ")

    ' Calcular tiempo en horas
    Dim horasEstacionamiento As Integer = CInt(Math.Ceiling((horaSalida - horaEntrada).TotalHours))

    ' Calcular monto a pagar
    Dim montoTotal As Decimal = CalcularMonto(horasEstacionamiento)

    ' Mostrar resumen de la transacción
    Console.WriteLine()
    Console.WriteLine("====== Resumen de la Transacción ======")
    Console.WriteLine("Nombre del Cliente: " & nombre)
    Console.WriteLine("Identificación del Vehículo: " & placa)
    Console.WriteLine("Tiempo en el parqueo: " & horasEstacionamiento & " horas")
    Console.WriteLine("Monto Total a Pagar: Q" & montoTotal.ToString("F2"))

    Console.WriteLine("Tiempo en el parqueo: " & horasEstacionamiento & " horas")
    Console.WriteLine("Monto Total a Pagar: Q" & montoTotal.ToString("F2"))

    ' Escribir la factura en el archivo de texto
    EscribirFactura(nombre, nit, placa, horasEstacionamiento, montoTotal)
End Sub

End Module
```

D. Resultado



E. Diagrama de Flujo y Algoritmo

Algoritmo para el programa de estacionamiento

1. **Inicio del programa**
   - Mostrar un mensaje de bienvenida.
2. **Solicitar y validar datos del cliente**:
   - Solicitar el nombre del cliente.
   - Solicitar el NIT del cliente.
   - Solicitar la identificación del vehículo (número de placa).
3. **Solicitar y validar las horas de entrada y salida**:
   - Solicitar la hora de entrada en formato "HH
   ".
   - Validar si el formato de la hora de entrada es correcto.
   - Solicitar la hora de salida en formato "HH
   ".
   - Validar si el formato de la hora de salida es correcto.
4. **Calcular el tiempo total de estacionamiento**:
   - Restar la hora de entrada de la hora de salida.
   - Convertir la diferencia de tiempo en horas.
   - Redondear el tiempo total hacia arriba si hay fracciones de hora (usando Math.Ceiling).
5. **Calcular el monto total a pagar**:
   - Si el tiempo total es 1 hora o menos, el costo es Q15.00.
   - Si es más de 1 hora, cobrar Q15.00 por la primera hora y Q20.00 por cada hora adicional.
6. **Mostrar resumen de la transacción**:
   - Mostrar el nombre del cliente.
   - Mostrar la identificación del vehículo.
   - Mostrar la cantidad de horas de estacionamiento.
   - Mostrar el monto total a pagar.
7. **Generar y guardar la factura**:
   - Abrir o crear el archivo facturas.txt.
   - Escribir los detalles de la transacción en el archivo: nombre del cliente, NIT, placa, tiempo de estacionamiento, y monto total.
   - Guardar y cerrar el archivo.
8. **Finalizar**:
   - Mostrar un mensaje de éxito al usuario.
   - Terminar el programa.

```
Inicio -> Solicitar Nombre, NIT, Placa -> Solicitar Hora Entrada y Salida
    |
Validar horas <- Si error, volver a solicitar horas
    |
Calcular Tiempo de Estacionamiento -> Calcular Monto Total
    |
Mostrar Resumen -> Generar Factura (facturas.txt)
    |
Mostrar Mensaje de Éxito -> Fin
```

F. Conclusiones

1. **Eficiencia en el Cálculo de Tarifas**: El programa desarrollado cumple con el objetivo de calcular de manera precisa y eficiente el monto a pagar por el servicio de estacionamiento, utilizando una estructura de tarifas simple y clara. Esto asegura que los usuarios obtengan un cálculo justo y exacto basado en su tiempo de uso, lo cual es fundamental para la satisfacción del cliente y la transparencia en la operación del servicio.
2. **Robustez y Fiabilidad del Programa**: A través de la implementación de funciones de validación de datos, el programa demuestra ser robusto y fiable, minimizando la posibilidad de errores por entradas incorrectas. Esto asegura que el sistema maneje adecuadamente los posibles errores de usuario, contribuyendo a una experiencia de usuario más fluida y sin interrupciones.
3. **Efectividad en la Generación y Almacenamiento de Facturas**: El programa logra generar y almacenar correctamente las facturas de las transacciones en un archivo de texto (facturas.txt). La información registrada es clara y organizada, lo que facilita futuras consultas y auditorías. Esto cumple con el objetivo de garantizar la persistencia y accesibilidad de los datos, lo cual es crucial para la gestión y control del servicio de estacionamiento.