UNIVERSIDAD AMERICANA
Facultad de Ingeniería y Arquitectura FIA

Documentación Código

C#

Estudiante:

Ervin David Pérez Hernández .

Alejandro Abdul Zarruk Sanchez.

Docente:

Silvia López

Documentación del Código Ejercicio#1

Estructura del Proyecto

El proyecto se llama `TrabajoPar_1` y está compuesto por dos clases principales: `Program` y `Metodos`. La clase `Program` contiene el punto de entrada del programa, mientras que la clase `Metodos` se encarga de la lógica para gestionar la información de las personas.

1. Clase `Program`

```
internal class Program

{

      static void Main(string[] args)

      {

       Metodos metodos = new Metodos();

        metodos.menu();

        Console.ReadKey();

      }

}
```

- **Método `Main`**: Es el punto de entrada del programa. Aquí se crea una instancia de la clase `Metodos` y se llama al método `menu`, que inicia la interacción con el usuario.

2. Clase `Metodos`

2.1. Estructura `Datos`

```
public struct Datos

{

      public string Nombre;

      public string Direccion;

      public int Telefono, Edad;


      public Datos(string nombre, string direccion, int telefono, int edad)

      {

        Nombre = nombre;
```

```
        Direccion = direccion;

        Telefono = telefono;

      Edad = edad;

      }
}
```

- **Estructura `Datos`**: Define una estructura que contiene información sobre una persona, incluyendo su nombre, dirección, teléfono y edad. Proporciona un constructor para inicializar estos campos.

2.2. Lista de Personas

```
private List<Datos> personas = new List<Datos>();
```

- **`personas`**: Una lista que almacena instancias de la estructura `Datos`, representando a cada persona ingresada.

2.3. Método `DatosPersonas`

```
public void DatosPersonas()

{

    Console.WriteLine("Ingrese el Nombre de la Persona ");

      string nombre = Console.ReadLine();

    Console.WriteLine("Ingrese la Edad de la Persona ");

      int edad = int.Parse(Console.ReadLine());

    Console.WriteLine("Ingrese el telefono de la Persona");

      int telefono = int.Parse(Console.ReadLine());

    Console.WriteLine("Ingrese la Direccion de la Persona ");

    string direccion = Console.ReadLine();


      Datos nuevaPersona = new Datos(nombre, direccion, telefono, edad);

    personas.Add(nuevaPersona);

}
```

- **Funcionalidad**: Solicita al usuario que ingrese los datos de una nueva persona, crea una nueva instancia de `Datos` y la añade a la lista `personas`.

2.4. Método `Mostrarnombres`

```
public void Mostrarnombres()

{

      if (personas.Count == 0)

      {

        Console.WriteLine("No hay Personas en La lista");

      return;

      }

    Console.WriteLine("Los nombres en la Lista son");

      foreach (var persona in personas)

      {

        Console.WriteLine(persona.Nombre);

      }

}
```

- **Funcionalidad**: Muestra todos los nombres de las personas en la lista. Si la lista está vacía, informa al usuario.

2.5. Método `MostrarporEdad`

```
public void MostrarporEdad(int edad)

{

    Console.WriteLine($"Persona de edad {edad} años");

      foreach (var persona in personas)

      {

      if (persona.Edad == edad)

      {

            Console.WriteLine($"{persona.Nombre}");

      }

      }

}
```

- **Funcionalidad**: Muestra los nombres de las personas que tienen una edad específica.

2.6. Método `MostrarPorNombre`

```
public void MostrarPorNombre(string nombre)

{

	Console.WriteLine($"persona con el nombre {nombre}");

	foreach (var persona in personas)

	{

	if (persona.Nombre == nombre)

	{

		Console.WriteLine($"{persona.Nombre}");

	}

	}

}
```

- **Funcionalidad**: Muestra los nombres de las personas que coinciden con un nombre específico ingresado por el usuario.

2.7. Método `menu`

```
public void menu()

{

	int opcion;

	do

	{

	Console.WriteLine("Menu");

	Console.WriteLine("1 .Mostrar Listas de los nombres");

	  Console.WriteLine("2 .Mostrar las Personas de una cierta Edad");

	  Console.WriteLine("3 .Mostrar la personas que coincidan los
nombres");

	  Console.WriteLine("4 .Agregar Persona");

	  Console.WriteLine("5 .Salir de el Programa");
```

```
  Console.WriteLine("Elige una Opcion ");
opcion = int.Parse(Console.ReadLine());
switch (opcion)
{
      case 1:
          Mostrarnombres();
          break;
      case 2:
          Console.WriteLine("Introduce la Edad de la persona ");
          int edad = int.Parse(Console.ReadLine());
          MostrarporEdad(edad);
          break;
      case 3:
          Console.WriteLine("Ingrese el Nombre ");
          string nombre = Console.ReadLine();
          MostrarPorNombre(nombre);
          break;
      case 4:
      if (personas.Count <= 9)
      {
              DatosPersonas();
          }
          else
          {
              Console.WriteLine("La lista esta Llena");
          }
          break;
      case 5:
```

```
                Console.WriteLine("Saliendo de el Programa..............");
                break;
            default:
                Console.WriteLine("Por favor escoja una opcion valida ");
                break;
        }
        } while (opcion != 5);
}
```

- **Funcionalidad**: Muestra un menú al usuario para interactuar con el programa. Permite mostrar los nombres, buscar por edad o nombre, agregar nuevas personas y salir del programa.

Ejercicio # 2

1. Definición de la clase Funciones

Se crea una clase Funciones que agrupa una clase estática interna llamada VentasNegocio. Esta clase VentasNegocio se encarga de manejar toda la lógica de las ventas.

```
internal class Funciones
{
      public static class VentasNegocio
      {
      // Funciones relacionadas con las ventas
      }
}
```

2. Método IngresarVentas

- Este método solicita al usuario que ingrese las ventas para cada día y las almacena en un arreglo.
- Utiliza un bucle for para recorrer el arreglo y pedir la venta de cada día.

```
public static void IngresarVentas(double[] ventas)

{

     for (int i = 0; i < ventas.Length; i++) // Recorre cada día
de la semana

     {

     Console.Write($"Ingrese las ventas del día {i + 1}: ");

     ventas[i] = Convert.ToDouble(Console.ReadLine()); // Lee y
almacena la venta

     }

}
```

Explicación

El parámetro ventas[] es un arreglo que almacenará las ventas para cada día. ventas.Length será de 7 porque es una semana (7 días).

Se utiliza el bucle for para solicitar al usuario que ingrese la venta de cada día.

En cada iteración, i + 1 representa el número de día para mostrar al usuario. Por ejemplo, para el día 1, se muestra "Ingrese las ventas del día 1".

Ejemplo: El usuario ingresa las siguientes ventas:

Día 1: 100

Día 2: 200

Día 3: 150

Día 4: 180

Día 5: 220

Día 6: 190

Día 7: 210

El arreglo ventas[] queda así:

ventas = [100, 200, 150, 180, 220, 190, 210]

3. Método CalcularTotalVendido

Este método suma todas las ventas de la semana y retorna el total.

```
public static double CalcularTotalVendido(double[] ventas)

{

      double total = 0; // Inicializa el total en 0



      foreach (var venta in ventas) // Recorre cada venta del
arreglo

      {

      total += venta; // Suma cada venta al total

      }



      return total; // Retorna el total

}
```

Explicación :

1. El parámetro ventas[] es el mismo arreglo que contiene las ventas de cada día.
2. Se inicializa una variable total en 0. Esta será usada para almacenar la suma de todas las ventas.
3. Se utiliza un bucle foreach para recorrer cada valor de ventas[] y sumar ese valor a total.
4. Al final, la función retorna el total de las ventas.

Ejemplo: Para el arreglo de ventas:
ventas = [100, 200, 150, 180, 220, 190, 210]

La suma sería:
100 + 200 + 150 + 180 + 220 + 190 + 210 = 1250

El total vendido es 1250.

4. Método EncontrarDiaMaxVenta

- Este método encuentra el día en el que se realizó la venta más alta y retorna el índice de ese día.

```
public static int EncontrarDiaMaxVenta(double[] ventas)

{

     int diaMax = 0; // Inicializa el día con la venta máxima como
el día 1 (índice 0)

     double maxVenta = ventas[0]; // Asume que la mayor venta es
la del primer día



     for (int i = 1; i < ventas.Length; i++) // Comienza desde el
segundo día (índice 1)

     {

     if (ventas[i] > maxVenta) // Si encuentra una venta mayor

     {

          maxVenta = ventas[i]; // Actualiza el valor de la mayor
venta

          diaMax = i; // Guarda el índice del día con la mayor
venta

     }

     }



     return diaMax; // Retorna el índice del día con la mayor
venta

}
```

Explicación:

1. Se inicializa la variable diaMax con 0, lo que indica que, inicialmente, se asume que el primer día (índice 0) tiene la mayor venta.
2. maxVenta se inicializa con la venta del primer día (ventas[0]).
3. El bucle for comienza en el segundo día (índice 1) y compara cada venta con maxVenta.
4. Si encuentra una venta mayor, actualiza maxVenta y el índice de diaMax al día correspondiente.
5. Al final, retorna el índice del día con la venta más alta.

Ejemplo: Dado el arreglo ventas = [100, 200, 150, 180, 220, 190, 210], la venta más alta es 220, que ocurre en el quinto día (índice 4).
La función retorna 4, pero en el menú se muestra 4 + 1 = 5 para indicar el día 5.

5. Clase Program y el Menú

- La clase Program contiene el método Main, que implementa un menú interactivo para que el usuario pueda ingresar ventas, calcular el total y ver el día con la venta más alta.

```
internal class Program

{

      static void Main(string[] args)

      {

      const int dias = 7; // Cantidad de días en la semana

      double[] ventas = new double[dias]; // Arreglo para almacenar
las ventas de cada día

      bool salir = false; // Variable para controlar si se desea
salir del programa



      while (!salir) // Bucle para mostrar el menú hasta que el
usuario elija salir

      {

            Console.WriteLine("Menu de Ventas");

            Console.WriteLine("1. Ingresar ventas");

            Console.WriteLine("2. Calcular total de ventas");

            Console.WriteLine("3. Mostrar día con la venta más
alta");

            Console.WriteLine("4. Salir");

            Console.Write("Ingrese una opción: ");

            string opcion = Console.ReadLine(); // Lee la opción
seleccionada por el usuario



            switch (opcion) // Control de las opciones del menú

            {
```

```
                case "1":

Funciones.VentasNegocio.IngresarVentas(ventas); // Llama al método
para ingresar las ventas
                    break;
                case "2":
                    double totalVendido =
Funciones.VentasNegocio.CalcularTotalVendido(ventas); // Calcula
el total
                    Console.WriteLine($"Total vendido:
{totalVendido}"); // Muestra el total
                    break;
                case "3":
                    int diaMaxVenta =
Funciones.VentasNegocio.EncontrarDiaMaxVenta(ventas); // Encuentra
el día con la venta más alta
                    Console.WriteLine($"Día con la venta más
alta: {diaMaxVenta + 1}"); // Muestra el día (se suma 1 para
ajustar el índice)
                    break;
                case "4":
                    salir = true; // Termina el programa
                    break;
                default:
                    Console.WriteLine("Opción no válida, intente
de nuevo."); // Mensaje de error para opciones no válidas
                    break;
            }

            Console.WriteLine();
        }
```

```
        }

}
```

Explicación:

1. Se define un arreglo ventas[] con 7 elementos para almacenar las ventas de cada día de la semana.
2. El menú se ejecuta dentro de un bucle while que continuará hasta que el usuario elija la opción 4 para salir.
3. Dependiendo de la opción elegida, el programa:
    - Opción 1: Llama al método IngresarVentas() para ingresar las ventas.
    - Opción 2: Calcula y muestra el total vendido con CalcularTotalVendido().
    - Opción 3: Muestra el día con la venta más alta utilizando EncontrarDiaMaxVenta().
    - Opción 4: Sale del programa.
4. Si el usuario ingresa una opción no válida, se muestra un mensaje de error.

Ejercicio # 3

1. Ingreso de una temperatura en Celsius

Cuando el programa se ejecuta, solicita al usuario que ingrese una temperatura en grados Celsius. Supongamos que el usuario ingresa 25 grados Celsius.

Fragmento del código:

```
Console.WriteLine("Ingrese una temperatura en grados Celsius:");

double tempCelsius;

if (double.TryParse(Console.ReadLine(), out tempCelsius))

{

      Funciones.AgregarTemperatura(tempCelsius);

    Funciones.ConvertirTemperaturas(tempCelsius);

}

else
```

```
{
      Console.WriteLine("Por favor ingrese un número válido.");
}
```

Explicación:

- El programa muestra el mensaje "Ingrese una temperatura en grados Celsius:".
- El usuario ingresa el valor 25, que es convertido en un valor de tipo double y almacenado en la variable tempCelsius.
- Se llama a la función AgregarTemperatura(tempCelsius), que añade la temperatura a la lista de temperaturas en Celsius.
- Luego, el programa llama a ConvertirTemperaturas(tempCelsius), que convierte el valor ingresado a Fahrenheit y Kelvin, mostrando los resultados.

Resultado en consola

```
Ingrese una temperatura en grados Celsius:
25
La temperatura 25 °C es equivalente a 77 °F y 298.15 K.
```

2. Conversión de temperaturas

La función ConvertirTemperaturas realiza la conversión de la temperatura ingresada (en grados Celsius) a las otras dos escalas: **Fahrenheit** y **Kelvin**.

Fragmento del código:

```
public static void ConvertirTemperaturas(double celsius)
{
      double fahrenheit = CelsiusAFahrenheit(celsius);
      double kelvin = CelsiusAKelvin(celsius);

      temperaturasFahrenheit.Add(fahrenheit);
```

```
        temperaturasKelvin.Add(kelvin);


        Console.WriteLine($"La temperatura {celsius} °C es equivalente a
{fahrenheit} °F y {kelvin} K.");

}
```

Explicación:

- La función recibe como parámetro la temperatura en Celsius y llama a dos funciones auxiliares: CelsiusAFahrenheit y CelsiusAKelvin, que realizan las conversiones.
- **CelsiusAFahrenheit** convierte los grados Celsius a Fahrenheit con la fórmula:

```
return (celsius * 9 / 5) + 32;
```

· **CelsiusAKelvin** convierte los grados Celsius a Kelvin con la fórmula:

```
return celsius + 273.15;
```

- Las temperaturas convertidas se almacenan en las listas temperaturasFahrenheit y temperaturasKelvin.
- Finalmente, el programa muestra los resultados de las conversiones en la consola.

Ejemplo de conversión (para 25°C):

```
La temperatura 25 °C es equivalente a 77 °F y 298.15 K.
```

3. Mostrar todas las listas de temperaturas

El programa puede mostrar todas las temperaturas almacenadas (en grados Celsius, Fahrenheit y Kelvin) en listas separadas. Esto sucede después de cada ingreso de temperatura.

Fragmento del código:

```
public static void MostrarListas()

{

    Console.WriteLine("\nTemperaturas en Celsius:");

    MostrarLista(temperaturasCelsius);

    Console.WriteLine("Temperaturas en Fahrenheit:");

    MostrarLista(temperaturasFahrenheit);

    Console.WriteLine("Temperaturas en Kelvin:");

    MostrarLista(temperaturasKelvin);

}
```

Explicación:

- La función MostrarListas llama a otra función auxiliar MostrarLista, que recibe una lista de temperaturas y las imprime con su índice.
- El resultado es una presentación clara de todas las temperaturas en las tres escalas. Aquí hay un ejemplo de cómo se ve cuando hay varias entradas:

Supongamos que el usuario ingresó las siguientes temperaturas: 25°C, 30°C y 15°C.

Resultado en consola:

```
Temperaturas en Celsius:

1. 25

2. 30

3. 15


Temperaturas en Fahrenheit:

1. 77

2. 86

3. 59
```

```
Temperaturas en Kelvin:

1. 298.15

2. 303.15

3. 288.15
```

4. Eliminar una temperatura de las listas

El usuario puede eliminar una temperatura de las listas de Celsius, Fahrenheit y Kelvin. El programa solicita al usuario que seleccione el número de la temperatura que desea eliminar.

Fragmento del código:

```
public static void EliminarTemperatura()

{

    Console.WriteLine("Seleccione el número de la temperatura que desea
eliminar de las listas:");

    MostrarLista(temperaturasCelsius);



      int indice;

      if (int.TryParse(Console.ReadLine(), out indice) && indice > 0 &&
indice <= temperaturasCelsius.Count)

      {

        temperaturasCelsius.RemoveAt(indice - 1);

        temperaturasFahrenheit.RemoveAt(indice - 1);

        temperaturasKelvin.RemoveAt(indice - 1);



        Console.WriteLine("Temperatura eliminada exitosamente.");

      }

      else

      {

        Console.WriteLine("Número inválido.");
```

```
        }
}
```

Explicación:

- El programa muestra todas las temperaturas en Celsius para que el usuario elija cuál desea eliminar.
- El usuario ingresa el número correspondiente, y si es válido, se elimina la temperatura en la posición seleccionada de las tres listas (Celsius, Fahrenheit, Kelvin).
- Si el número ingresado es incorrecto o fuera de rango, se muestra un mensaje de error.

Ejemplo de eliminación:

Supongamos que se han ingresado tres temperaturas y el usuario decide eliminar la segunda temperatura (30°C).

Consola antes de eliminar:

```
Seleccione el número de la temperatura que desea eliminar de las listas:
1. 25
2. 30
3. 15
```

El usuario ingresa 2. Las listas se actualizan automáticamente.

Consola después de eliminar:

```
Temperatura eliminada exitosamente.

Temperaturas en Celsius:
1. 25
2. 15
```

```
Temperaturas en Fahrenheit:

1. 77

2. 59


Temperaturas en Kelvin:

1. 298.15

2. 288.15
```

5. Repetición del programa y finalización

El programa permite ingresar nuevas temperaturas y seguir operando mientras el usuario lo desee. Al final de cada ciclo, pregunta si el usuario desea agregar otra temperatura.

Fragmento del código:

```
Console.WriteLine("¿Desea agregar otra temperatura? (s/n)");

continuar = Console.ReadLine().ToLower() == "s";
```

Explicación:

- Si el usuario responde con "s" (sí), el programa continúa y vuelve a solicitar una nueva temperatura.
- Si el usuario responde con "n" (no), el ciclo se rompe y el programa finaliza mostrando el mensaje "Programa finalizado.".

Ejemplo:

```
¿Desea agregar otra temperatura? (s/n)

n

Programa finalizado.
```

Ejercicio # 4

El proyecto está compuesto por dos clases principales: `Program` y `procesos`. Esta última gestiona un inventario de productos, permitiendo al usuario agregar productos y calcular el valor total del inventario.

1. Clase `Program`

```
class Program

{

      static void Main(string[] args)

      {

        procesos procesos = new procesos();

        procesos.ImprimirDatos();

        Console.ReadKey();

      }

}
```

- **Método `Main`: Punto de entrada del programa. Crea una instancia de `procesos` y llama al método `ImprimirDatos` para iniciar la gestión del inventario.**

2. Clase `procesos`

2.1. Estructura `Producto`

```
struct Producto

{

      public string Nombre;

      public float Precio;


      public Producto(string nombre, float precio)
```

```
        {
          Nombre = nombre;
          Precio = precio;
        }

        public string Imprimirdatos()
        {
        return $"{Nombre}: ${Precio}";
        }
}
```

- **Estructura `Producto`: Define un producto con nombre y precio. Incluye un método para imprimir los datos del producto.**

2.2. Método `CalcularValorTotal`

```
static float CalcularValorTotal(List<Producto> inventario)
{
        float total = 0;
        foreach (var producto in inventario)
        {
        total += producto.Precio;
        }
        return total;
}
```

- **Funcionalidad: Calcula el valor total de todos los productos en el inventario sumando sus precios.**

2.3. Método `ImprimirDatos`

```
public void ImprimirDatos()
{
        string NombreProducto;
```

```
      float PrecioProducto;
      int Variablecontrol = 0;
    List<Producto> Inventario = new List<Producto>();

      do
      {
        Console.WriteLine("Ingrese el Nombre de el Producto");
      NombreProducto = Console.ReadLine();
        Console.WriteLine("Ingrese el Precio de ese producto");
        PrecioProducto = float.Parse(Console.ReadLine());
        Inventario.Add(new Producto(NombreProducto, PrecioProducto));

        Console.WriteLine("Productos en el Inventario");
      foreach (var Producto in Inventario)
      {
            Console.Write(Producto.Imprimirdatos());
            Console.WriteLine();
      }

        Console.WriteLine("Desea agregar Otro producto al inventario 0.SI
1.NO");
        Variablecontrol = int.Parse(Console.ReadLine());
      } while (Variablecontrol == 0);

      // Calcular y mostrar el valor total del inventario
      float valorTotal = CalcularValorTotal(Inventario);
    Console.WriteLine($"Valor total del inventario: ${valorTotal}");
    Console.WriteLine("Gracias por usar el inventario.");
```

```
    Console.ReadKey();

}
```

- **Funcionalidad: Permite al usuario ingresar productos (nombre y precio) en un bucle hasta que decida no agregar más. Muestra todos los productos en el inventario y calcula el valor total al final.**