Universidad ORT Uruguay
Facultad de Ingeniería
Escuela de Tecnología

OBLIGATORIO PROGRAMACIÓN 2

Taramasco, Sabrina – 180403

Villar, Florencia - 218320

Zanetti, Florencia – 283076

Grupo M2A, 2022

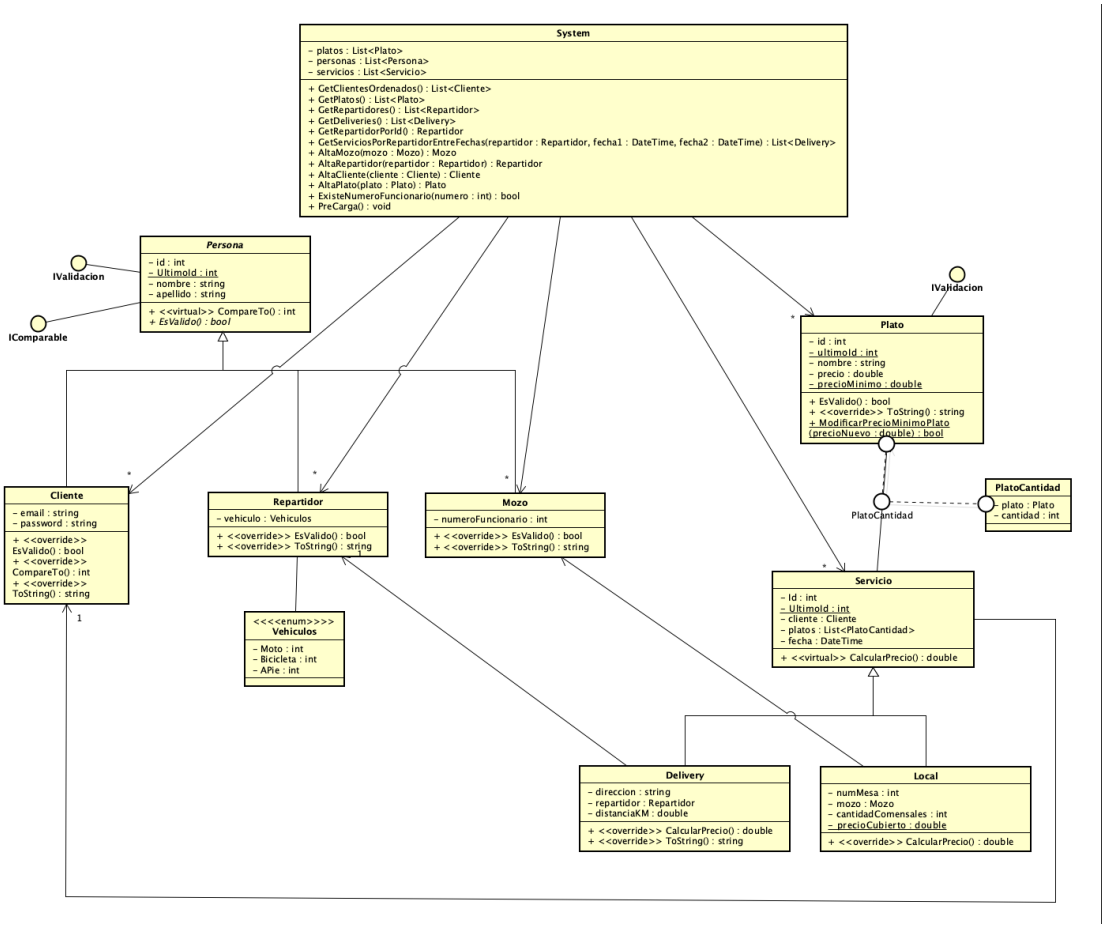
Docente: Rodriguez, Joaquin

Analista en Tecnologías de la información - Analista Programador

05/05/2022

Índice

Diagrama de clase del dominio del Problema



Tablas información precargada

Clase Persona	Clase Mozo (m1)	Clase Mozo (m2)	Clase Mozo (m3)	Clase Mozo (m4)	Clase Mozo (m5)
Nombre	Pedro	Leandro	Lorena	Paola	Santiago
Apellido	Fagundez	Sanchez	Varela	Pacheco	Benitez
NroFuncionario	1	2	3	4	5

Clase Persona	Clase Repartidor (r1)	Clase Repartidor (r2)	Clase Repartidor (r3)	Clase Repartidor (r4)	Clase Repartidor (r5)
Nombre	Pedro	Leandro	Lorena	Paola	Santiago
Apellido	Fagundez	Sanchez	Varela	Pacheco	Benitez
Vehiculo	APie	Bicicleta	Moto	Moto	APie

Clase Persona	Clase Cliente (c1)	Clase Cliente (c2)	Clase Cliente (c3)	Clase Cliente (c4)	Clase Cliente (c5)
Nombre	Juan	Romina	Claudia	Facundo	Florencia
Apellido	Gonzalez	Lopez	Pereira	Moreira	Martinez
Email	juan19@gmail.com	romina19@gmail.com	claudia19@gmail.com	facundo19@gmail.com	florencia19@gmail.com
Password	Juan3456	Romina3456	Claudia3456	Facundo3456	Florencia3456

Clase Servicio	Clase Local (l1)	Clase Local (l2)	Clase Local (l3)	Clase Local (l4)	Clase Local (l5)
Cliente	c1	c3	c2	c5	c5
Fecha	2021-03-03	2022-07-15	2020-10-19	2021-12-01	2021-08-23
NumeroMesa	1	3	4	2	4
Mozo	m4	m5	m1	m3	m2
CantidadComensales	7	2	1	5	4

Clase Servicio	Clase Delivery (d1)	Clase Delivery (d2)	Clase Delivery (d3)	Clase Delivery (d4)	Clase Delivery (d5)
Cliente	c1	c2	c3	c4	c5
Fecha	2022-03-03	2021-03-03	2021-04-03	2021-09-23	2022-01-30
Direccion	Ramirez 123	Bolivar 456	Bolivia 789	Nin y Silva 123	Rivera 456
Mozo	r1	r2	r3	r4	r5
PrecioCubierto	12	5	9	22	11

Clase Plato	Plato (p1)	Plato (p2)	Plato (p3)	Plato (p4)	Plato (p5)
Nombre	Milanesa	Pollo	Ensalada	Hamburguesa al plato	Nuggets
Precio	345	405	250	180	250

Clase Plato	Plato (p5)	Plato (p5)	Plato (p5)	Plato (p5)	Plato (p5)
Nombre	Guiso	Nioqui	Pascualina	Canelones de carne	Lasagna
Precio	450	370	180	270	410

PROGRAM

```
using System;
using System.Collections.Generic;
namespace ObligatorioP2
{
   class Program
   {
      static void Main(string[] args)
      {
         System s = new System(); //creamos la instancia de Sistema
         int op = -1; //variable op = -1 para que entre en el while.
         while (op != 0) //si presionamos 0, sale del programa.
         {
             MostrarMenu(); //método que muestra en consola las opciones de Menu.
              op = Int32.Parse(Console.ReadLine()); //lee la opcion del menu que selecciona el usuario.
            switch (op)
            {
               case 1:
                  Console.WriteLine("Listar platos");
```

```
                List<Plato> listaPlatos = s.GetPlatos(); //creamos la lista de platos llamando al método en System
                if (listaPlatos.Count > 0)
                {
                    foreach (Plato p in listaPlatos)//se recorren todos los platos para mostrarlos en consola.
                    {
                        Console.WriteLine(p);
                    }
                }
                else
                {
                    Console.WriteLine("No hay registros");
                }

                Console.ReadKey();
                break;
            case 2:
                List<Cliente> ListaOrdenada = s.GetClientesOrdenados(); //creamos la lista ordenada de clientes llamando al método en System
                if (ListaOrdenada.Count > 0)
                {
                    foreach (Cliente c in ListaOrdenada)
                    {
                        Console.WriteLine(c);
```

```
                }
            }
            else
            {
                Console.WriteLine("No hay registros");
            }
            Console.ReadKey();
            break;
        case 3:
                //almacenamos en variables el ID del Repartidor y dos fechas entre las que se buscan servicios de delivery.
            Console.WriteLine("Ingrese ID del Repartidor");
            int num = Int32.Parse(Console.ReadLine());
            Console.WriteLine("Ingrese fecha 1(AAAA-MM-DD)");
            DateTime fecha1 = DateTime.Parse(Console.ReadLine());
            Console.WriteLine("Ingrese fecha 2(AAAA-MM-DD)");
            DateTime fecha2 = DateTime.Parse(Console.ReadLine());
                                    List<Delivery> listaDeliverys = s.GetServiciosPorRepartidorEntreFechas(s.GetRepartidorPorId(num), fecha1, fecha2);
                //creamos la lista de deliverys llamando al método en System
            if (listaDeliverys.Count > 0)
            {

                foreach (Delivery d in listaDeliverys)
```

```
            {
                Console.WriteLine(d.ToString());
            }
        }
        else
        {
            Console.WriteLine("No hay registros");
        }
        Console.ReadKey();



        break;


    case 4:


                double precioMinimoActual = Plato.PrecioMinimo; //mostramos al usuario cual es el precio minimo actual del plato.

                Console.WriteLine($"El precio mínimo actual es: {precioMinimoActual}");


                Console.WriteLine("Ingrese nuevo precio mínimo del plato");

        double precioNuevo = Double.Parse(Console.ReadLine());

                bool precioCambiado = Plato.ModificarPrecioMinimoPlato(precioNuevo); //se modifica el precio del plato llamando al método en el System.
```

```
                if (precioCambiado)
                {
                        Console.WriteLine($"El nuevo precio mínimo es :
{precioNuevo}");

                }
                else
                {
                   Console.WriteLine("No se cambia el precio");
                }
                Console.ReadKey();



                break;

            case 5:
                //pedimos datos para registrar (dar de alta) a un mozo.
                Console.WriteLine("Ingrese el nombre del Mozo");
                string nombre = Console.ReadLine();

                Console.WriteLine("Ingrese el apellido del Mozo");
                string apellido = Console.ReadLine();
```

```
                Console.WriteLine("Ingrese número de funcionario");
                int nroFuncionario = Int32.Parse(Console.ReadLine());

                Mozo m = new Mozo(nombre, apellido, nroFuncionario);

                 if (s.AltaMozo(m) != null) //si se cumple el AltaMozo, este se muestra en la pantalla.
                {
                   Console.WriteLine($"El nuevo mozo es: {m}");

                }
                else
                {
                        Console.WriteLine("El mozo no se puede registrar - verifique que los campos se hayan completado o cambie el número de funcionario");
                }
                Console.ReadKey();
                break;
            }
        }
        if (op == 0)
        {
            Console.Clear(); //cerramos la consola presionando 0.
```

```
        }

    }

     private static void MostrarMenu()//método para mostrar Menu en pantalla.

    {

        Console.Clear();

        Console.WriteLine("### SISTEMA RESTAURANTE ###");

        Console.WriteLine("1 - Listar Platos");

                    Console.WriteLine("2 - Listar Clientes Ordenados por apellido/nombre");

          Console.WriteLine("3 - Listar servicios dado un repartidor y un rango de fechas");

        Console.WriteLine("4 - Cambiar precio mínimo del plato");

        Console.WriteLine("5 - Dar de alta un mozo");

        Console.WriteLine("0 - Salir");



    }


  }

}
```

SYSTEM

```
using System;
using System.Collections.Generic;



namespace ObligatorioP2
{
    public class System
    {



        //creamos las variables que contienen los diferentes tipos de listas
        //inicializamos las listas.
        private List<Plato> platos = new List<Plato>();
        private List<Persona> personas = new List<Persona>();
        private List<Servicio> servicios = new List<Servicio>();


        public System()
        {
            PreCarga(); //colocamos PreCarga en constructor de System.
        }


         public List<Cliente> GetClientesOrdenados() //método que retorna una lista de clientes ordenados por un criterio.
```

```
    {
        List<Cliente> clientesOrdenados = new List<Cliente>(); //creamos variable de tipo lista para los clientes ordenados.
        foreach (Persona p in personas)//recorremos la lista de personas.
        {
            if (p is Cliente)//toma aquellas personas que sean de tipo cliente.
            {
                Cliente aux = p as Cliente; //creamos una variable auxiliar Cliente(de la persona por el cual se encuentra el recorrido).
                clientesOrdenados.Add(aux); //agregamos el cliente auxiliar a una lista.
            }
        }
        clientesOrdenados.Sort(); //se ordena la lista mediante el uso del Sort.
        return clientesOrdenados; //devolvemos la lista ordenada.
    }

    public List<Mozo> GetMozos()
    {
        //el método (al igual que GetClientesOrdenados) devuelve una lista de mozos, obtenida de la lista personas.
        List<Mozo> ret = new List<Mozo>();
        foreach (Persona p in personas)
        {
            if (p is Mozo)
```

```
            {
                Mozo aux = p as Mozo;
                ret.Add(aux);
            }
        }

        return ret;
    }

    public List<Repartidor> GetRepartidores()
    {
        //el método (al igual que GetClientesOrdenados) devuelve una lista de repartidores, obtenida de la lista personas.
        List<Repartidor> ret = new List<Repartidor>();
        foreach (Persona p in personas)
        {
            if (p is Repartidor)
            {
                Repartidor aux = p as Repartidor;
                ret.Add(aux);
            }
        }

        return ret;
    }
```

```
public List<Delivery> GetDeliveries()
 //el método (similar que GetClientesOrdenados) devuelve una lista de deliveries, obtenida de la lista servicios.
{
   List<Delivery> ret = new List<Delivery>();
   foreach (Servicio s in servicios)
   {
      if (s is Delivery)
      {
         Delivery aux = s as Delivery;
         ret.Add(aux);
      }
   }

   return ret;
}

//retornamos la lista de platos.
public List<Plato> GetPlatos()
{
   return platos;
}

public Repartidor GetRepartidorPorId(int num)
```

```
{
    Repartidor ret = null;
    List<Repartidor> repartidores = GetRepartidores(); //creamos una lista de repartidores con GetRepartidores.

    foreach (Repartidor r in repartidores)
    {
            if(r.Id.Equals(num)) //si el repartidor tiene el ID igual al argumento num, se retorna el objwto repartidor.
      {
        ret = r;
      }
    }

    return ret;
}
```

```
public List<Delivery> GetServiciosPorRepartidorEntreFechas(Repartidor repartidor, DateTime fecha1, DateTime fecha2)

{
    List<Delivery> ret = new List<Delivery>();
```

```
            List<Delivery> deliverys = GetDeliveries(); //creamos lista de deliveries con GetDeliveries.


          //se recorre la lista de deliveries. Si el repartidor es el mismo que el argumento dado, y la fecha está en el rango
        //brindado: se agrega el objeto delivery a la lista de retorno.


        foreach (Delivery d in deliverys)
        {
            if (repartidor == d.Repartidor)
            {
                if (d.Fecha >= fecha1 && d.Fecha <= fecha2)
                {
                    ret.Add(d);
                }
            }
        }


        return ret;
    }




    //public bool ModificarPrecioMinimoPlato(double precioNuevo)
    //{
```

```
//     if (precioNuevo != Plato.PrecioMinimo && precioNuevo>=0) //si el precio nuevo es validado, se cambia el precio minimo de la clase Plato.
//   {
//   Plato.PrecioMinimo = precioNuevo;
//   return true;


//   }
//   return false;
//}


public Mozo AltaMozo(Mozo m)
{
   Mozo nuevo = null;
                                                  if (m.EsValido() && !NumeroFuncionarioExiste(m.NumeroFuncionario))
                //si los datos del funcionario son válidos y el número de funcionario no existe: se da de alta el nuevo Mozo.
   //agrega a Mozo a la lista de personas.
   {
      nuevo = m;
      personas.Add(nuevo);


   }
   return nuevo;
```

```
    }
    public Repartidor AltaRepartidor(Repartidor r)
    {
        Repartidor nuevo = null;
        if (r.EsValido())
            //si los datos del repartidor son válidos: se da de alta el nuevo
Repartidor.
        //agrega a Repartidor a la lista de personas.
        {
            nuevo = r;
            personas.Add(nuevo);


        }
        return nuevo;


    }
    public Plato AltaPlato(Plato p)
    {
        if (p.EsValido())
        //si los datos del plato son válidos: se da de alta el nuevo Plato.
        //agrega a Plato a la lista de platos.
        {
            platos.Add(p);
            return p;
        }
```

```
        return null;
    }

    public Cliente AltaCliente(Cliente c)
    {
        if (c.EsValido())
            //si los datos del cliente son válidos: se da de alta el nuevo Cliente.
        //agrega a Cliente a la lista de personas.
        {
            personas.Add(c);
            return c;
        }
        return null;
    }

    public bool NumeroFuncionarioExiste(int numero)
    {
        bool numExiste = false;
          foreach (Persona m in personas) //se recorren las personas, si la persona es Mozo se hace una variable auxiliar de tipo Mozo.

        {
            if (m is Mozo)
            {
```

```
            Mozo aux = m as Mozo;


                    if (aux.NumeroFuncionario == numero) //se verifica si el
numero de funcionario del mozo es igual al argumento.
            {
               numExiste = true;
            }


          }
        }


        return numExiste;
      }


        private void PreCarga()
        {
                            Cliente c1 = new Cliente("Juan", "Gonzalez",
"juan19@gmail.com", "Juan3456");
            AltaCliente(c1); //llamamos al método AltaCliente para validr y
agregar los clientes al sistema.


                            Cliente c2 = new Cliente("Romina", "Lopez",
"romina19@gmail.com", "Romina3456");
          AltaCliente(c2);
```

Cliente c3 = new Cliente("Claudia", "Pereira", "claudia19@gmail.com", "Claudia3456");

AltaCliente(c3);

Cliente c4 = new Cliente("Facundo", "Moreira", "facundo19@gmail.com", "Facundo3456");

AltaCliente(c4);

Cliente c5 = new Cliente("Florencia", "Martinez", "florencia19@gmail.com", "Florencia3456");

AltaCliente(c5);

Mozo m1 = new Mozo("Pedro", "Fagundez", 1);

AltaMozo(m1); //llamamos al método AltaMozo para validr y agregar los mozos al sistema.

Mozo m2 = new Mozo("Leandro", "Sanchez", 2);

AltaMozo(m2);

Mozo m3 = new Mozo("Lorena", "Varela", 3);

AltaMozo(m3);

Mozo m4 = new Mozo("Paola", "Pacheco", 4);

AltaMozo(m4);

Mozo m5 = new Mozo("Santiago", "Benitez", 5);

AltaMozo(m5);

Repartidor r1 = new Repartidor("Javier", "Perez", Repartidor.Vehiculos.APie);

AltaRepartidor(r1); //llamamos al método AltaRepartidor para validr y agregar los repartidores al sistema.

Repartidor r2 = new Repartidor("Gonzalo", "Ramirez", Repartidor.Vehiculos.Bicicleta);

AltaRepartidor(r2);

Repartidor r3 = new Repartidor("Maria", "Gutierrez", Repartidor.Vehiculos.Moto);

AltaRepartidor(r3);

Repartidor r4 = new Repartidor("Rossana", "Villar", Repartidor.Vehiculos.Moto);

AltaRepartidor(r4);

Repartidor r5 = new Repartidor("Ana", "Cubero", Repartidor.Vehiculos.APie);

AltaRepartidor(r5);

Delivery d1 = new Delivery(c1, DateTime.Parse("2022-03-03"), "Ramirez 123", r1, 12);

servicios.Add(d1); //no es relevante la validación de los datos, se agregan directamente al sistema.

Delivery d2 = new Delivery(c2, DateTime.Parse("2021-03-03"), "Bolivar 456", r2, 5);

servicios.Add(d2);

```
            Delivery d3 = new Delivery(c3, DateTime.Parse("2021-04-03"),
"Bolivia 789", r3, 9);

            servicios.Add(d3);

            Delivery d4 = new Delivery(c4, DateTime.Parse("2021-09-23"),
"Nin y Silva 123", r4, 22);

            servicios.Add(d4);

            Delivery d5 = new Delivery(c5, DateTime.Parse("2022-01-30"),
"Rivera 456", r5, 11);

            servicios.Add(d5);

            Local l1 = new Local(c1, DateTime.Parse("2021-03-03"), 1, m4, 7);

            servicios.Add(l1);

            Local l2 = new Local(c3, DateTime.Parse("2022-07-15"), 3, m5, 2);

            servicios.Add(l2);

            Local l3 = new Local(c2, DateTime.Parse("2020-10-19"), 4, m1, 1);

            servicios.Add(l3);

            Local l4 = new Local(c5, DateTime.Parse("2021-12-01"), 2, m3, 5);

            servicios.Add(l4);

            Local l5 = new Local(c5, DateTime.Parse("2021-08-23"), 4, m2,
4);

            servicios.Add(l5);



            Plato p1 = new Plato("Milanesa", 345);

                AltaPlato(p1); //llamamos al método AltaPlato para validr y
agregar los platos al sistema.
```

```
Plato p2 = new Plato("Pollo", 405);
AltaPlato(p2);

Plato p3 = new Plato("Ensalada", 250);
AltaPlato(p3);

Plato p4 = new Plato("Hamburguesa al plato", 180);
AltaPlato(p4);

Plato p5 = new Plato("Nuggets", 250);
AltaPlato(p5);

Plato p6 = new Plato("Guiso", 450);
AltaPlato(p6);

Plato p7 = new Plato("Nioqui", 370);
AltaPlato(p7);

Plato p8 = new Plato("Pascualina", 180);
AltaPlato(p8);

Plato p9 = new Plato("Canelones de carne", 270);
AltaPlato(p9);
```

```
            Plato p10 = new Plato("Lasagna", 410);
            AltaPlato(p10);
        }
    }
}
```

PERSONA

```
using System;
using System.Diagnostics.CodeAnalysis;

namespace ObligatorioP2
{
    public abstract class Persona : IValidacion, IComparable<Persona>
    {
        // creamos las instancias que tienen en común todas las clases hijas de clase Persona.
        public int Id { get; set; }

        public static int UltimoId { get; set; }

        public string Nombre { get; set; }
```

```
public string Apellido { get; set; }



public Persona()
{
}

public Persona(string nombre, string apellido)
{
    Id = UltimoId;
    UltimoId++;
    Nombre = nombre;
    Apellido = apellido;
}

 public virtual int CompareTo([AllowNull] Persona other) // le damos un criterio para saber como es un apellido respecto de otro.
                                                         // ese criterio retorna un numero: 1 si es mayor a 0, -1 si es menor a 0.
                                                        // luego el Sort se encarga de ordenar a los objetos por apellido.

{
        if(Apellido.CompareTo(other.Apellido) > 0)       // el criterio se aplica primero segun los apellidos y luego por nombres.
    {
```

```
            return 1;
        }
        else if(Apellido.CompareTo(other.Apellido) < 0)
        {
            return -1;
        }
        else
        {
            if(Nombre.CompareTo(other.Nombre) > 0)
            {
                return 1;
            }
            else if(Nombre.CompareTo(other.Nombre) < 0)
            {
                return -1;
            }
            else
            {
                return 0;
            }
        }
    }
```

public abstract bool EsValido(); //declaramos el método EsValido (abtracto en una clase abstracta), no lleva porción de código.

```
                                //esta clase padre(Persona) no sabe como la clase hija lo va a resolver.
                                //luego es definida y utilizado por las clases hijas.



    }
}
```

CLIENTE

```
using System;
using System.Diagnostics.CodeAnalysis;

namespace ObligatorioP2
{
    public class Cliente : Persona
    {

        public string Email { get; set; }

        public string Password { get; set; }
```

```
        public Cliente()
        {
        }

           public Cliente(string nombre, string apellido, string email, string
password)
        {
            Id = UltimoId;
            UltimoId++;
            Nombre = nombre;
            Apellido = apellido;
            Email = email;
            Password = password;
        }

        public override bool EsValido()
        {
            bool stringTieneNumero = false;
            bool esValidoCliente = false;
            bool hayArroba = false;
            bool hayNum = false;
            bool hayMayus = false;
            bool hayMinus = false;
```

```
if (Nombre != "" && Apellido != "")
{

    for (int i = 0; i < Nombre.Length; i++)
    {
        char caracter = Nombre[i];
        if (Char.IsNumber(caracter))
        {
            stringTieneNumero = true;
        }
    }

    for (int i = 0; i < Apellido.Length; i++)
    {
        char caracter2 = Apellido[i];
        if (Char.IsNumber(caracter2))
        {
            stringTieneNumero = true;
        }
    }

}

for (int i = 1; i < Email.Length - 1; i++)
```

```
{
    char caracter = Email[i];
    if (Char.ToString(caracter) == "@")
    {
        hayArroba = true;
    }
}

if (Password.Length >= 6)
{
    for (int i = 0; i < Password.Length; i++)
    {
        char caracter = Password[i];
        if (Char.IsNumber(caracter))
        {
            hayNum = true;
        }
        else
        {
            if (Char.ToUpper(caracter) == caracter)
            {
                hayMayus = true;
            }
            if (Char.ToLower(caracter) == caracter)
```

```
                {
                    hayMinus = true;
                }

            }

        }

    }

            if (hayArroba && hayNum && hayMinus && hayMayus &&
!stringTieneNumero)
    {
        esValidoCliente = true;
    }

    return esValidoCliente;
}

public override int CompareTo([AllowNull] Persona other)
{
    return base.CompareTo(other);
}

public override string ToString()
```

```
        {
            return $"{Apellido},{Nombre}";
        }
    }
}
```

REPARTIDOR

```
using System;

namespace ObligatorioP2

{

    public class Repartidor : Persona // la clase Repartidor hereda instancias y métodos de su clase padre (Persona).

  {

    // creamos las instancias unicas de las clase Repartidor

    public Vehiculos Vehiculo { get; set; }

      public enum Vehiculos //utilizamos la función Enum para definir los distintos tipos de vehículos

    {

      Moto,

      Bicicleta,

      APie,

    }


    public Repartidor()

    {

    }


        public Repartidor(string nombre, string apellido, Vehiculos vehiculo)

    {

      Id = UltimoId;
```

```
        UltimoId++;
        Nombre = nombre;
        Apellido = apellido;
        Vehiculo = vehiculo;
    }

        public override bool EsValido() //la clase Repartidor hereda la función EsValido de su clase padre (Persona)
                                        //sobreescribe la función realizando sus propias validaciones
    {
        bool esValido = false;

        if (Nombre != "" && Apellido != "") //valida que nombre y apellido no estén vacíos
        {
            esValido = true;

            for (int i = 0; i < Nombre.Length; i++) //valida que nombre no contenga números.

            {
                char caracter = Nombre[i];      //busca en los caracteres del nombre si existe o no un número.
                if (Char.IsNumber(caracter))
                {
```

```
                esValido = false;
            }
        }

            for (int i = 0; i < Apellido.Length; i++) //valida que apellido no contenga números.

        {
            char caracter2 = Apellido[i];      //busca en los caracteres del apellido si existe o no un número.
            if (Char.IsNumber(caracter2))
            {
                esValido = false;
            }
        }


    }
    return esValido;
}

    public override string ToString() //función que retorna un objeto en formato string, y lo representa como una cadena de caracteres.
    {
```

```
            return $"{Apellido},{Nombre}"; //muestra en la consola el Apellido
y el Nombre.


        }

    }

}
```

MOZO

```
using System;

namespace ObligatorioP2

{

    public class Mozo : Persona // la clase Mozo hereda instancias y métodos de su clase padre (Persona).

  {



      // creamos las instancias unicas de las clase Mozo

      public int NumeroFuncionario { get; set; }




      public Mozo()

      {

      }


                  public Mozo(string nombre, string apellido, int numeroFuncionario)

      {

         Id = UltimoId;

         UltimoId++;

         Nombre = nombre;

         Apellido = apellido;

         NumeroFuncionario = numeroFuncionario;
```

```
    }


        public override bool EsValido() //la clase Mozo hereda el método EsValido de su clase padre (Persona)

                                        //sobreescribe el método realizando sus propias validaciones

    {

        bool esValido = false;



          if (Nombre != "" && Apellido != "") //valida que nombre y apellido no estén vacíos

        {

            esValido = true;



              for (int i = 0; i < Nombre.Length; i++)//valida que nombre no contenga números.



            {

                  char caracter = Nombre[i];      //busca en los caracteres del nombre si existe o no un número.

                if (Char.IsNumber(caracter))

                {

                    esValido = false;

                }

            }
```

```
                for (int i = 0; i < Apellido.Length; i++)//valida que apellido no contenga números.


            {

                    char caracter2 = Apellido[i]; //busca en los caracteres del apellido si existe o no un número.

                if (Char.IsNumber(caracter2))

                {

                    esValido = false;

                }

            }



          }

          return esValido;

      }


         public override string ToString()  //función que retorna un objeto en formato string, y lo representa como una cadena de caracteres.

      {


                return $"{Apellido}, {Nombre}. Su número de funcionario es: {NumeroFuncionario}";   //muestra en la consola el Apellido, el Nombre y el Número de Funcionario.


      }
```

```
    }
}
```

SERVICIO

```
using System;
using System.Collections.Generic;

namespace ObligatorioP2
{
    public abstract class Servicio // si existe una instancia de servicio, ejemplo take away, no seria abstract.
    {

        // creamos las instancias que tienen en común todas las clases hijas de clase Servicio.
        public int Id { get; set; }

        public static int UltimoId { get; set; }

        public Cliente Cliente { get; set; }

        private List<PlatoCantidad> platos = new List<PlatoCantidad>();

        public DateTime Fecha { get; set; }



        public Servicio()
        {
```

```
}

public Servicio(Cliente cliente, DateTime fecha)
{
    Id = UltimoId;
    UltimoId++;
    Cliente = cliente;
    Fecha = fecha;
}
```

```
public virtual double CalcularPrecio() //la clase padre Servicio emplea CalcularPrecio a su manera.
                                       //las clases hijas la utilizan o no, y lo resuelven como lo hizo su clase padre.
{
    double sumaMontos = 0;

foreach (PlatoCantidad pc in platos) //para cada plato de la lista que contiene la cantidad de los distintos plato, me quedo con su precio y la cantidad.

    {
        sumaMontos += pc.Plato.Precio * pc.Cantidad; //calculamos el precio total (sumamos todos los platos).
```

```
            }
            return sumaMontos;
        }



    }
}
```

DELIVERY

```
using System;

namespace ObligatorioP2
{
    public class Delivery : Servicio // la clase Delivery hereda instancias y métodos de su clase padre (Servicio).
    {
        //creamos las instancias únicas de la clase Delivery
        public string Dirección { get; set; }

        public Repartidor Repartidor { get; set; }

        public double DistanciaKM { get; set; }
```

```
public Delivery()
{
}
```

```
public Delivery(Cliente cliente, DateTime fecha, string dirección, Repartidor repartidor, int distanciaKM)
{
    Id = UltimoId;
    UltimoId++;
    Cliente = cliente;
    Fecha = fecha;
    Dirección = dirección;
    Repartidor = repartidor;
    DistanciaKM = distanciaKM;
}
```

```
public override double CalcularPrecio() //redefine la función CalcularPrecio traída de su clase padre y utiliza el resto.
{
                double sumaMontos = base.CalcularPrecio(); //base.CalcularPrecio() es la parte de la función que trae desde la clase padre Servicio.
                //precio base.

    double costoEnvio; //redefine la función.
```

```
            if (DistanciaKM < 2) //en distancias menores a 2km tiene un
costo de envío de 50 uy.
        {
            costoEnvio = 50;
        }
        else
        {
             double distancia = Math.Round(DistanciaKM); //distancias que
sean de 2km o más.
                                                    //utilizamos Math.Round para
redondear un valor doble al valor entero más cercano, ya que va
aumentando 10 pesos por km.

             double costoExtraDistancia = distancia * 10 - 10; //calculamos el
costo extra según aumenten los kms.

            costoEnvio = costoExtraDistancia;
        }

        if(sumaMontos > 100) //aumenta hasta un máximo de 100 pesos.
                                //una vez que supere los 100 pesos, se mantiene el
precio(100).
        {
            costoEnvio = 100;
        }
```

```
            return sumaMontos + costoEnvio;

        }


        public override string ToString() //función que retorna un objeto en formato string, y lo representa como una cadena de caracteres.

        {

            return $"Servicio para: {Cliente.Nombre}, en la fecha: {Fecha}, el repartidor fue: {Repartidor.Nombre}."; //muestra en la consola el Nombre del cliente y la Fecha del envío.

        }

    }

}
```

LOCAL

```
using System;

namespace ObligatorioP2

{

    public class Local : Servicio // la clase Local hereda instancias y métodos de su clase padre (Servicio).

  {

    //creamos las instancias únicas de la clase Local.

    public int NumeroMesa { get; set; }


    public Mozo Mozo { get; set; }


    public int CantidadComensales { get; set; }


    public static double PrecioCubierto { get; set; }


    public Local()

    {

    }


        public Local(Cliente cliente, DateTime fecha, int numeroMesa, Mozo mozo, int cantidadComensales)

    {

       Id = UltimoId;
```

```
            UltimoId++;
            Cliente = cliente;
            Fecha = fecha;
            NumeroMesa = numeroMesa;
            Mozo = mozo;
            CantidadComensales = cantidadComensales;
        }

            public override double CalcularPrecio() //redefine la función CalcularPrecio traída de su clase padre y utiliza el resto.
        {
                                double sumaMontos = base.CalcularPrecio(); //base.CalcularPrecio() es la parte de la función que trae desde la clase padre Servicio.

            double propina = sumaMontos * 0.1; //redefine la función.
                                    //se agrega un 10% de propina

                                sumaMontos += propina + PrecioCubierto * CantidadComensales; //se agrega el precio del cubierto

            return sumaMontos;
        }
    }
}
```

PLATO

```
using System;
namespace ObligatorioP2
{
    public class Plato: IValidacion
    {
        public int Id { get; set; }

        public static int UltimoId { get; set; }

        public string Nombre { get; set; }

        public double Precio { get; set; }

        public static double PrecioMinimo { get; set; } = 100;


        public Plato()
        {
        }

        public Plato(string nombre, double precio)
        {
            Id = UltimoId;
```

```
    UltimoId++;
    Nombre = nombre;
    Precio = precio;
}



public bool EsValido()
{
    bool nombreEsValido = false;
    bool precioEsValido = false;
    bool esValido = false;

    if (Nombre != "")
    {
        nombreEsValido = true;

        for (int i = 0; i < Nombre.Length; i++)
        {
            char caracter = Nombre[i];
            if (Char.IsNumber(caracter))
            {
                nombreEsValido = false;
            }
```

```
            }


        }
        if (Precio >= PrecioMinimo)
        {
            precioEsValido = true;
        }


        if(precioEsValido && nombreEsValido)
        {
            esValido = true;
        }
        return esValido;
    }
                public static bool ModificarPrecioMinimoPlato(double precioNuevo)
    {
            if (precioNuevo != PrecioMinimo && precioNuevo >= 0) //si el precio nuevo es validado, se cambia el precio minimo de la clase Plato.
        {
            PrecioMinimo = precioNuevo;
            return true;


        }
        return false;
```

```
        }


        public override string ToString()
        {
            return $"{Nombre}: {Precio}.   ";
        }


    }


}
```

PLATO CANTIDADusing System;

namespace ObligatorioP2

{

public class PlatoCantidad //creamos una clase de asociación (PlatoCantidad) porque un comensal puede llevar muchos platos y puede repetir un plato

//esto hace que haya el mismo plato en una lista.

{

//creamos las instancias únicas de la clase de asociación PlatoCantidad.

```
        public Plato Plato { get; set; }


        public int Cantidad { get; set; }
```

```
        public PlatoCantidad()
        {
        }

        public PlatoCantidad(Plato plato, int cantidad)
        {

            Plato = plato;
            Cantidad = cantidad;
        }
    }
}
```

IVALIDACIÓN

```
using System;

namespace ObligatorioP2
{
```

interface IValidacion // define los métodos de validación de datos que deben implementar los objetos.

```
{

    //declaramos el método para que sea utilizado.

    public bool EsValido();









}

}
```