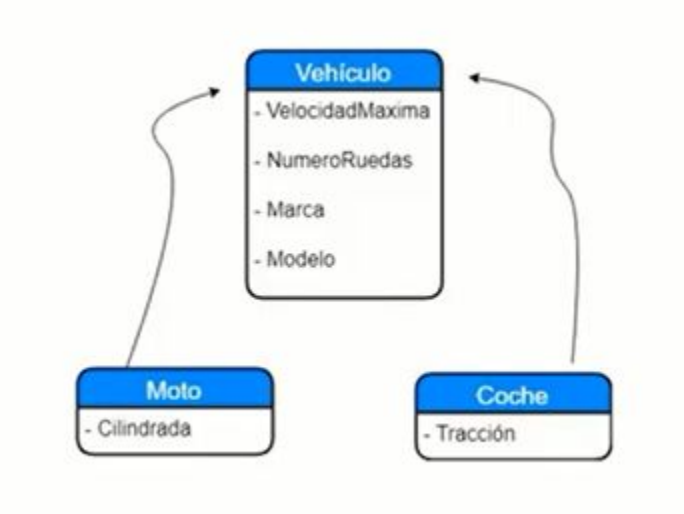
Programación Intermedia

Herencia


Vehículo
- VelocidadMaxima
- NumeroRuedas
- Marca
- Modelo
Moto
- Cilindrada
Coche
- Tracción

Polimorfismo

- En tiempo de ejecución (interfaces)
- Estático
  - New
  - Virtual/override

Encapsulamiento y Abstracción

Encapsulación: Ligada a los modificadores de acceso:

- public
- private
- internal
- protected
- protected internal
- private protected

Abstracción: nos permite representar el mundo real de una manera más simple.

Interfaz

Contrato entre dos entidades

- métodos
- propiedades
- indexers
- events

Sobrecarga de métodos

Métodos con el mismo nombre pero diferentes parámetros

- cantidad de parámetros
- tipo de los parámetros
- orden de los parámetros

Principios SOLID

Principio de Responsabilidad Unica

“Una clase debe tener una sola razón para cambiar”

```
3 references
public void GuardarArticulo(string contenido, string titulo)
{
    Log.Information($"vamos a insertar el articulo {titulo}");
    File.WriteAllText($"{path}/{titulo}.txt", contenido);
    Log.Information($"articulo {titulo} insertado");
    this.Cache.Add(titulo, contenido);

}

4 references
public string ConsultarArticulo(string titulo)
{
    Log.Information($"Consultar artículo {titulo}");

    string contenido = this.Cache.Get(titulo);
    if (!string.IsNullOrWhiteSpace(contenido))
    {
        Log.Information($"Artículo encontrado en la cache {titulo}");
        return contenido;
    }

    Log.Information($"buscar articulo en el sistema de archivos {titulo}");
    contenido = File.ReadAllText($"{path}/{titulo}.txt");

    return contenido;
}
```

```
2 references
public class Logging
{
    5 references
    public void Info(string message)
    {
        Log.Information(message);
    }
    0 references
    public void Error(string message, Exception e)
    {
        Log.Error(e, message);
    }
    0 references
    public void Fatal(string message, Exception e)
    {
        Log.Fatal(e, message);
    }
}
```

```
public class ArticulosServicio
{

    private readonly Logging _logging;

    public ArticulosServicio()
    {
        _logging = new Logging();
    }

    public void GuardarArticulo(string contenido, string titulo)
    {
        _logging.Info($"vamos a insertar el articulo {titulo}");
        File.WriteAllText($"{path}/{titulo}.txt", contenido);
        _logging.Info($"articulo {titulo} insertado");
        this.Cache.Add(titulo, contenido);
    }
    //REsto del código
}
```

```
Using Serilog;
public class Logging
{
    public void Info(string message)
    {
        Log.Information(message);
    }
    public void Error(string message, Exception e)
    {
        Log.Error(e, message);
    }
    public void Fatal(string message, Exception e)
    {
        Log.Fatal(e, message);
    }
}
```

```
using Log4Net;
public class Logging
{
    public void Info(string message)
    {
        Log.Info(message);
    }
    public void Error(string message, Exception e)
    {
        Log.Error(message, e);
    }
    public void Fatal(string message, Exception e)
    {
        Log.Fatal(message, e);
    }
}
```

Practicar con Almacenamiento

Principio Abierto/Cerrado

“Abierta para su extensión, Cerrada para su modificación”

Ejemplo:

- Herencia
- Composición

Practicar con guardar Log en BD

Principio de Sustitución de Liskov

“Subtipos pueden ser sustituidos por sus tipos base”

Violacion del principio:

- NotImplementedException();
- Extracción de interfaces

Practicar con guardar Almacenamiento en BD

Principio de Segregación de Interfaces

“Los clientes no deben ser forzados a depender de métodos que no utilizan”

Diferencia con liskov: Liskov dice que una clase puede ser reemplazada por otra sin notar la diferencia, en cambio la segregación de interfaces dice que un interfaz debe ser creada sabiendo que todas las clases que la implementen no van a estar forzadas a implementar alguno de sus métodos.

Practicar con guardar Almacenamiento en Archivos

Inversión de Dependencias

El principio establece:

- los módulos de alto nivel no deberían depender de los módulos de bajo nivel. Ambos deberían depender de abstracciones.

Listas List<T>

- Permite almacenar una colección de datos
- No hay límite de tamaño
- Mutables
- Permiten ejecutar consultas LINQ

```
List<string> ejemploLista = new List<string>();

ejemploLista.Add(“nueva”);

ejemploLista.Add(“lista”);
```

Algunos métodos de las listas

```
string[] ejemploCiudadesArray =  ejemploListaCiudades.ToArray();
ejemploListaCiudades.AddRange(ejemploListaCiudades2);
int numeroItems = ejemploListaCiudades.Count;
string primeraCiudad = ejemploListaCiudades.First();
string ultimaCiudad = ejemploListaCiudades.Last();
ejemploListaCiudades.Clear();
int numeroItemsDespuesDeVaciar = ejemploListaCiudades.Count;
```

LINQ

Lenguaje de consultas

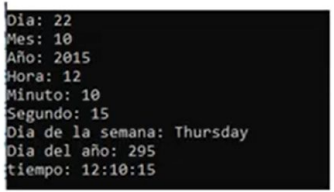
```
IEnumerable<string> ciudadesEncontradas = ejemploListaCiudades.Where(x => x.ToLower().Contains("e"));
foreach (var ciudad in ciudadesEncontradas)
{
    Console.WriteLine(ciudad);
}
```

DateTime

```
public DateTime(long ticks);
public DateTime(long ticks, DateTimeKind kind);
public DateTime(int year, int month, int day);
public DateTime(int year, int month, int day, Calendar calendar);
public DateTime(int year, int month, int day, int hour, int minute, int second);
public DateTime(int year, int month, int day, int hour, int minute, int second, DateTimeKind kind);
public DateTime(int year, int month, int day, int hour, int minute, int second, Calendar calendar);
public DateTime(int year, int month, int day, int hour, int minute, int second, int millisecond);
public DateTime(int year, int month, int day, int hour, int minute, int second, int millisecond, DateTimeKind kind);
public DateTime(int year, int month, int day, int hour, int minute, int second, int millisecond, Calendar calendar);
public DateTime(int year, int month, int day, int hour, int minute, int second, int millisecond, Calendar calendar, DateTimeKind kind);
```

Propiedades de DateTime

```
Console.WriteLine("Dia: " + ejemploFecha.Day);
Console.WriteLine("Mes: " + ejemploFecha.Month);
Console.WriteLine("Año: " + ejemploFecha.Year);
Console.WriteLine("Hora: " + ejemploFecha.Hour);
Console.WriteLine("Minuto: " + ejemploFecha.Minute);
Console.WriteLine("Segundo: " + ejemploFecha.Second);
Console.WriteLine("Dia de la semana: " + ejemploFecha.DayOfWeek);
Console.WriteLine("Dia del año: " + ejemploFecha.DayOfYear);
Console.WriteLine("tiempo: " + ejemploFecha.Date);
```



Métodos de DateTime

```
ejemploFecha.AddDays(1);
ejemploFecha.AddMonths(1);
ejemploFecha.AddYears(1);
ejemploFecha.AddDays(-1);
ejemploFecha.ToString();
ejemploFecha.ToShortDateString();
```

Añadir tiempo usando TimeSpan

```
...public TimeSpan(long ticks);
...public TimeSpan(int hours, int minutes, int seconds);
...public TimeSpan(int days, int hours, int minutes, int seconds);
...public TimeSpan(int days, int hours, int minutes, int seconds, int milliseconds);
```

Comparación de Fechas

```
int comparacion = DateTime.Compare(fecha, fechaActualizada);

int comparacion = fecha.CompareTo(fechaActualizada);

if (comparacion < 0)
{
    Console.WriteLine("La primera fecha es anterior");
}
else if (comparacion == 0)
{
    Console.WriteLine("es la misma fecha");
}
else
{
    Console.WriteLine("La segunda fecha es anterior");
}
```

Imprimir la fecha con formato

```
ejemploFecha.ToString();
ejemploFecha.ToShortDateString();
ejemploFecha.ToLongDateString();
ejemploFecha.ToShortTimeString();
```

```
22/02/2019 12:10:15
22/02/2019
Friday 22 February 2019
12:10
```

```
ejemploFecha.ToString("d");
ejemploFecha.ToString("D");
ejemploFecha.ToString("t");
```

```
22/02/2019 12:10:15
22/02/2019
Friday 22 February 2019
12:10
```

```
siMulacionFechaSistema.ToString("yyyy-MM-ddThh:mm:ss.ms")
```

```
2019-01-06T05:30:12.3012
```

Parámetros por Valor y por Referencia en Métodos

Por valor: mandamos una copia del valor de esa variable al método.

Por referencia:

- mandamos la posición de memoria del parámetro
  - ref
  - out
  - in

Tipos Anónimos

Un tipo anónimo es una clase que no tiene nombre, lo cual quiere decir que no tenemos esa clase como tal en el código. La gran mayoría de las veces, utilizaremos tipos anónimos cuando realizamos queries.

Lo único que necesitamos es utilizar la palabra reservada `new`

```
var equipo = new { Nombre = "Real Betis", Ligas = 1 };
```

Tipos Nullables

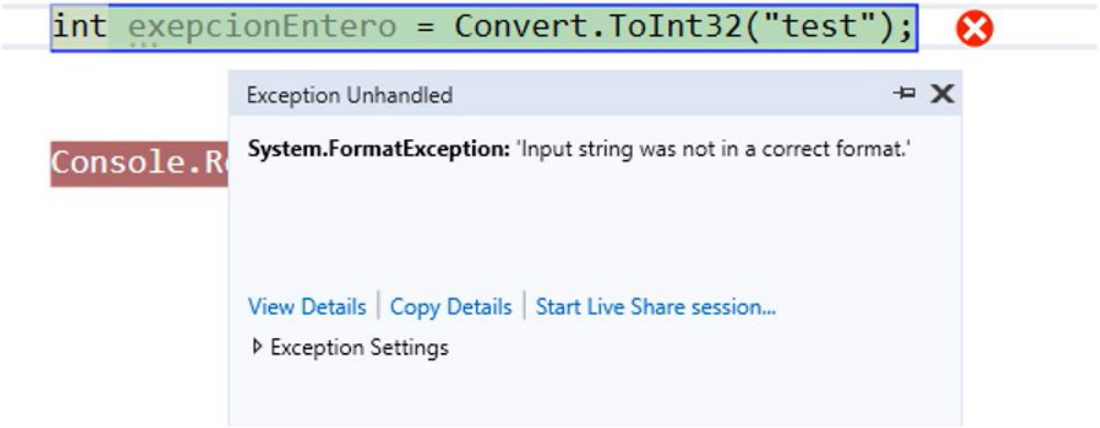
¿Qué es un tipo nullable?

- Es un tipo de dato que podemos inicializar a null

Manejo de Excepciones

- Que es una excepción
- Controlar las excepciones
- Controlar múltiples excepciones
- Crear Excepciones

Que es una excepción



Controlar las Excepciones

Bloque Try-Catch

```
try
{
    int numeroParaDividir = Convert.ToInt32(numeroEnteroIntroducido);
    Console.WriteLine($"El resultado de la division es: {25 / numeroParaDividir}");
}
catch (Exception e)
{
    Console.WriteLine($"Ha habido un error en la ejecución del programa: {e.Message.ToString()}");
}
```

Múltiples Excepciones

```
try
{
    int numberParaDividir = Convert.ToInt32(numeroEnteroIntroducido);
    Console.WriteLine($"El resultado de la division es: {25 / numberParaDividir}");
}
catch (FormatException e)
{
    Console.WriteLine($"El texto introducido no era un numero");
}
catch(DivideByZeroException e)
{
    Console.WriteLine($"Dividir por cero no esta permitido");
}
catch(Exception e)
{
    Console.WriteLine($"Ha habido un error en la ejecuacion del programa: {e.Message.ToString()}");
}
```

Creación de Excepciones en C#

Crear una excepción

Utilizar nuestra propia excepción