Trabalho Prático – POO

LESI – PL

Programação Orientada a Objetos

Diogo Gomes Silva – 23893

Barcelos, Braga 04/12/2022

Índice:

1. Introdução:

No âmbito da Unidade Curricular de Programação Orientada a Objetos integrada no 1º semestre do 2º ano do curso de LESI e visa o reforço e a aplicação dos conhecimentos adquiridos ao longo do semestre no âmbito de programação em linguagem C#. Com este trabalho prático pretende-se que sejam desenvolvidas soluções em C# para problemas reais de complexidade moderada. Serão identificadas as principais entidades (classes) envolvidas, as estruturas para suportar os dados e implementados os principais processos capazes de suportar soluções para esses problemas.

2. Propósitos e Objetivos:

OBJETIVOS GERAIS:

• Consolidar conceitos basilares do Paradigma Orientado a Objetos;

• Analisar problemas reais;

• Desenvolver capacidades de programação em C#;

• Potenciar a experiência no desenvolvimento de software;

• Assimilar o conteúdo da Unidade Curricular.

OBJETIVOS CONCRETOS:

• Saber identificar e implementar classes e objetos num problema real;

• Saber produzir código com qualidade, bem documentado, de acordo com a norma CLS;

• Saber gerar a API com a documentação do código que produziu (Doxigen ou outro);

• Saber aplicar os pilares da POO: Herança, Encapsulamento, Abstração e Polimorfismo;

• Saber estruturar devidamente uma solução em bibliotecas de classes;

• Saber estruturar uma solução por camadas, seguindo padrões como NTier, MVC, ou outro;

• Conseguir definir uma estratégia de persistência de dados;

• Saber analisar o código que produziu;

• Saber executar testes simples sobre o código produzido;

• Saber explorar das vantagens que o C# oferece.

OUTROS OBJETIVOS:

• Saber utilizar ferramentas de gestão de versões de código.

• Saber utilizar ferramentas de gestão de projetos.

3. Tema:

MOBILIDADE

O IPCA pretende oferecer um novo conceito de mobilidade usando as bicicletas Cabi do projeto U_Bike Portugal:

• As bicicletas estarão disponíveis numa das biciboxes, instaladas no Campus do IPCA;

• A bicibox irá dispor de um sistema EasyPark para bloqueio das bicicletas;

• A utilização das bicicletas terá um custo, dependendo do tipo de bicicleta e do tempo de utilização.

▪Os SAS são responsáveis pela manutenção das bicicletas e por manter o estado/disponibilidade destas atualizado.

•Os utilizadores devem proceder à reserva da bicicleta, sendo registados os momentos de levantamento e devolução da bicicleta.

• A solução a implementar deverá incluir ainda as seguintes funcionalidades:

• Gestão do saldo e pagamentos;

• Aplicação de coimas por devoluções fora de prazo ou danos na bicicleta.

4. Desenvolvimento das Classes (Fase1):

IDENTIFICAÇÃO DAS CLASSES:

Para a resolução do problema proposto foi definida (até ao momento) a necessidade de criação de 9 classes:

- Pessoa: servirá de partida para fazer as classes Utilizador, Funcionário e Mecânico através de heranças;

- Utilizador: Classe herdada da classe Pessoa que irá conter as informações respetivas aos utilizadores do programa;

- Funcionário: Classe herdada da classe Pessoa que irá conter as informações respetivas aos funcionários do programa;

- Mecânico: Classe herdada da classe Pessoa que irá conter as informações respetivas aos mecânicos do programa;

- Veículo: Irá conter as informações respetivas aos veículos do programa;

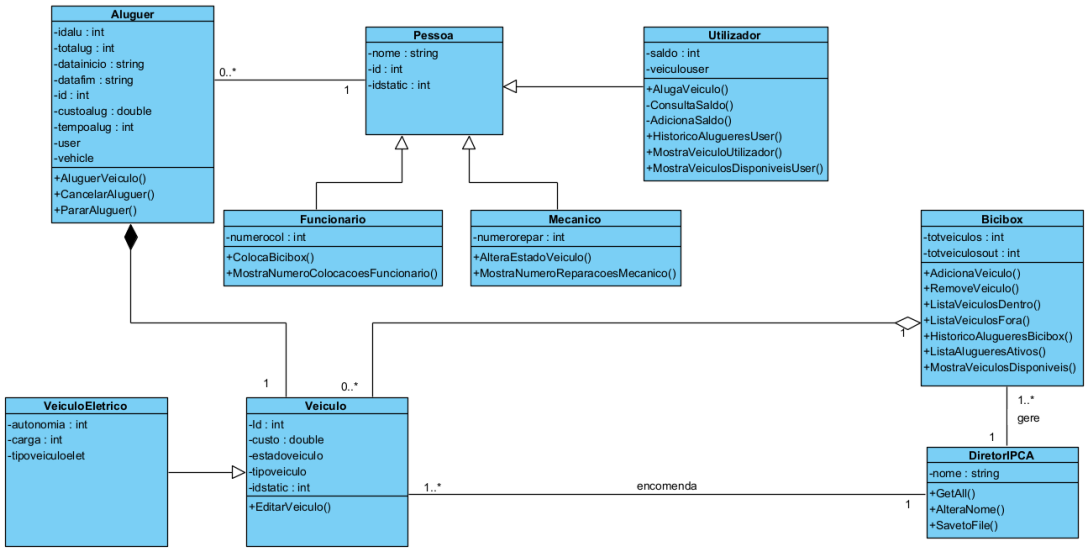
- VeiculoEletrico: Classe herdada da classe Veiculo que irá conter as informações respetivas aos Veiculos Elétricos do programa;

- Bicibox: Classe estática que irá conter as informações respetivas à Bicibox do programa, esta será responsável pela devida organização dos veículos;

- Aluguer: Irá conter as informações respetivas aos Alugueres de veículos efetuados pelos Utilizadores;

- DiretorIPCA: Irá conter a lista de Biciboxes do IPCA e permitirá ao Diretor aceder a todas as informações disponibilizadas pelo programa;

DIAGRAMA DE CLASSES:



DESCRIÇÃO DE ATRIBUTOS DAS CLASSES:

Pessoa:

- nome: Nome da Pessoa;

- id: Numero de identificação da Pessoa;

- idstatic: id estático da classe Pessoa.

Utilizador:

- saldo: Saldo do Utilizador (em euros);

- veiculouser: Veiculo que se encontra na posse do Utilizador.

Funcionário:

- numerocol: Numero de colocações de veículos na bicibox efetuadas pelo Funcionário.

Mecânico:

- numerorepar: Numero de reparações de veículos na bicibox efetuadas pelo Mecânico.

Veículo:

- id: Numero de identificação do veiculo;

- custo: Valor do custo do veiculo (por minuto);

- estadoveiculo: Estado em que se encontra o veículo (Disponível, Ocupado ou Avariado);

- idstatic: id estático da classe Veiculo;

- tipoveiculo: Tipo de veículo (Bicicleta).

VeículoEletrico:

- autonomia: Tempo de autonomia do veículo (minutos);

- carga: Carga elétrica que o veiculo tem (em percentagem);

- tipoveiculoelet: Tipo de veículo elétrico (Bicicleta elétrica ou Trotinete elétrica).

Aluguer:

- idalu: id estático da classe Veiculo;

- totalug: Número total de alugueres;

- datainicio: Data de início do aluguer;

- datafim: Data de fim do aluguer;

- id: Número de identificação do aluguer;

- custoalug: Custo total do Aluguer;

- tempoalug: Tempo de duração do Aluguer (em minutos);

- user: Utilizador que alugou o Veiculo;

- vehicle: Veiculo alugado pelo Utilizador.

Bicibox:

- totveiculos: Número total de veículos contidos na Bicibox;

- totveiculosout: Número total de veículos que se encontram fora da Bicibox (em utilização ou abandonados);

DiretorIPCA:

- nome: Nome do Diretor.

DESENVOLVIMENTO DAS CLASSES EM CÓDIGO:

Pessoa:

Começamos por criar a classe Pessoa no código atribuído os seguintes Atributos:

```
3 referências
public class Pessoa
{
    #region ATRIBUTOS

    /// <summary>
    /// Atributos da classe Pessoa
    /// </summary>

    string nome;  //nome da pessoa
    int id = 0;       //id da pessoa
    static int idstatic = 0 ;   //id static da classe pessoas

    TIPOPESSOA tipopessoa;

    #endregion
```

Também foi criado um enumerado “TIPOPESSOA” para atribuir o tipo de pessoa:

```
8 referências
public enum TIPOPESSOA   //enumerado tipos de pessoas
{
    Utilizador,
    Funcionario,
    Mecanico
}
```

Definimos também as seguintes propriedades para os seus atributos:

```
#region PROPRIEDADES

/// <summary>
/// Propriedades dos atributos da classe Pessoa
/// Permitem aceder aos atributos privados através dos getters e setters (encapsulamento)
/// </summary>

12 referências
public string Nome { get { return nome; } set { nome = value; } }

9 referências
public int Id { get { return id; } set { id = value; } }

12 referências
public int Idstatic { get { return idstatic; } set { idstatic = value; } }

9 referências
public TIPOPESSOA Tipopessoa
{
    get { return tipopessoa; }
    set { tipopessoa = value; }
}

#endregion
```

Utilizando o princípio do encapsulamento, estas Propriedades permitem "esconder" os atributos e listas que definimos como privados.

Utilizador:

Para a classe Utilizador foram criados os seguintes atributos:

```
public class Utilizador : Pessoa
{
    #region ATRIBUTOS

    double saldo;    //saldo do utilizador

    Veiculo veiculouser;   //armazena o veiculo que o utilizador alugou

    #endregion
```

E as seguintes propriedades para eles:

```
#region PROPRIEDADES

/// <summary>
/// Propriedades dos atributos da classe Utilizador
/// Permitem aceder aos atributos privados através dos getters e setters (encapsulamento)
/// </summary>

8 referências
public double Saldo { get { return saldo; } set { saldo = value; } }

2 referências
public Veiculo Veiculouser
{
    get { return veiculouser; }
    set { veiculouser = value; }
}

#endregion
```

Foram também criados os construtores da classe Utilizador que nos permitem criar os objetos Utilizador:

```
#region CONSTRUTORES

/// <summary>
/// Construtores da classe Utilizador
/// Permitem criar um utilizador
/// Um dos construtores recebe argumentos e o outro não recebe (Principio do Polimorfismo)
/// </summary>

0 referências
public Utilizador()
{
    Idstatic++;
    Id = Idstatic;
    Nome = "Utilizador Indef";
    Saldo = 0;
    Tipopessoa = TIPOPESSOA.Utilizador;
}

3 referências
public Utilizador(string nome, int saldo)
{
    Idstatic++;
    Id = Idstatic;
    this.Nome = nome;
    this.saldo = saldo;
    Tipopessoa = TIPOPESSOA.Utilizador;
}

#endregion
```

Com a utilização do princípio de Polimorfismo criamos um construtor que não recebe nenhum parâmetro e define um utilizador padrão com atributos padrão.

E por ultimo foi criado um Override no To String da classe Utilizador que permite mostrar as informações de cada utilizador:

```
#region Overrides

/// <summary>
/// Overrides da classe Pessoa
/// Permitem mostrar as informacoes dos utilizadores na consola
/// </summary>

0 referências
public override string ToString()    //permite mostrar na consola as informacoes das Pessoas
{
    string outStr = String.Format("Id: {0}\t Nome: {1}\t Saldo: {2}\t Tipo: {3}\n", Id, Nome, saldo, Tipopessoa);
    return outStr;
}

#endregion
```

Funcionário:

Para o Funcionário foi criado um atributo assim como a sua Propriedade:

```
8 referências
public class Funcionario : Pessoa
{
    /// <summary>
    /// Atributos da classe Funcionario
    /// </summary>

    #region ATRIBUTOS

    int numerocol = 0;    //numero de veiculos colocados pelo funcionario na bicibox

    #endregion


    #region PROPRIEDADES


    /// <summary>
    /// Propriedades dos atributos da classe Funcionario
    /// Permitem aceder aos atributos privados atavés dos getters e setters (encapsulamento)
    /// </summary>

    3 referências
    public int Numerocol
    {
        get { return numerocol; }
        set { numerocol = value; }
    }

    #endregion
```

Os seus construtores:

```
#region CONSTRUTORES

0 referências
public Funcionario()
{
    Idstatic++;
    Id = Idstatic;
    Nome = "Funcionario Indef";
    Tipopessoa = TIPOPESSOA.Funcionario;
}

0 referências
public Funcionario(string nome)
{
    Idstatic++;
    Id = Idstatic;
    this.Nome = nome;
    Tipopessoa = TIPOPESSOA.Funcionario;
}

#endregion
```

E o seu Override do ToString:

```
#region Overrides

/// <summary>
/// Overrides da classe Mecanico
/// Permitem mostrar as informacoes dos funcionarios na consola
/// </summary>

0 referências
public override string ToString()    //permite mostrar na consola as informacoes dos funcionarios
{
    string outStr = String.Format("Id: {0}\t Nome: {1}\t Numero de Colocacoes: {2}\t Tipo: {3}\n", Id, Nome, Numerocol, Tipopessoa);
    return outStr;
}

#endregion
```

Mecânico:

Para a classe Mecânico foi criado um atributo e sua propriedade:

```
6 referências
public class Mecanico : Pessoa
{
    #region ATRIBUTOS

    /// <summary>
    /// Propriedades dos atributos da classe Funcionario
    /// Permitem aceder aos atributos privados através dos getters e setters (encapsulamento)
    /// </summary>

    int numerorepar = 0;

    #endregion


    #region PROPRIEDADES

    3 referências
    public int Numerorepar
    {
        get { return numerorepar; }
        set { numerorepar = value; }
    }

    #endregion
```

Os construtores:

```
#region CONSTRUTORES

0 referências
public Mecanico()
{
    Idstatic++;
    Id = Idstatic;
    Nome = "Mecanico Indef";
    Tipopessoa = TIPOPESSOA.Mecanico;
}

0 referências
public Mecanico(string nome)
{
    Idstatic++;
    Id = Idstatic;
    this.Nome = nome;
    Tipopessoa = TIPOPESSOA.Mecanico;
}

#endregion
```

E o Override do ToString:

```
#region Overrides

/// <summary>
/// Overrides da classe Mecanico
/// Permitem mostrar as informacoes dos utilizadores na consola
/// </summary>

0 referências
public override string ToString()    //permite mostrar na consola as informacoes dos Mecanicos
{
    string outStr = String.Format("Id: {0}\t Nome: {1}\t Numero de Reparacoes: {2}\t Tipo: {3}\n", Id, Nome, Numerorepar, Tipopessoa);
    return outStr;
}

#endregion
```

Veículo:

Na criação da classe Veículo definimos dois Enumerados fora desta classe que nos permitem definir o tipo de Veículo (bicicleta) e o seu estado (Disponível, Ocupado ou Avariado). Foram também definidos os atributos da classe:

```
/// <summary>
/// Enumerados da classe Veiculo
/// </summary>

[Serializable]
9 referências
public enum TIPOVEICULO   //enumerado dos tipos de veiculo
{
    Bicicleta,
}

[Serializable]
19 referências
public enum ESTADOVEICULO   //enumerado dos estados do veiculo
{
    Disponivel,
    Ocupado,
    Avariado
}

/// <summary>
/// Classe Veiculo
/// irá armazenar informações dos Veiculos
/// </summary>
```

```
public class Veiculo
{
    #region ATRIBUTOS

    /// <summary>
    /// Atributos da classe Veiculo
    /// </summary>

    int id ;           //numero de identificacao do veiculo
    double custo;      //valor do custo do veiculo
    static int idstatic = 0;    //id estatico da classe veiculo


    TIPOVEICULO tipoveiculo;           //tipo de veiculo
    ESTADOVEICULO estadoveiculo;       //estado do veiculo

    #endregion
```

Foram também definidas as Propriedades desta classe para os respetivos atributos:

```
#region PROPRIEDADES

/// <summary>
/// Propriedades dos atributos da classe Veiculo
/// Permitem aceder aos atributos privados atavés dos getters e setters (encapsulamento)
/// </summary>

10 referências
public int Id
{
    get { return id; }
    set { id = value; }
}

7 referências
public double Custo
{
    get { return custo; }
    set { custo = value; }
}

3 referências
public int Idstatic
{
    get { return idstatic; }
    set { idstatic = value; }
}

5 referências
public TIPOVEICULO Tipoveiculo
{
    get { return tipoveiculo; }
    set { tipoveiculo = value; }
}

14 referências
public ESTADOVEICULO Estadoveiculo
{
    get { return estadoveiculo; }
    set { estadoveiculo = value; }
}

#endregion
```

E os construtores da mesma:

```
#region CONSTRUTORES

/// <summary>
/// Construtores da classe Veiculo
/// Permitem criar um veiculo
/// Um dos construtores recebe argumentos e o outro não recebe (Principio do Polimorfismo)
/// </summary>

0 referências
public Veiculo()
{
    idstatic++;
    Id = idstatic;
    Tipoveiculo = TIPOVEICULO.Bicicleta;
    Estadoveiculo = ESTADOVEICULO.Disponivel;
    Custo = 0.2;
}

3 referências
public Veiculo(TIPOVEICULO tipo, ESTADOVEICULO estado, double custo)
{
    idstatic++;
    Id = idstatic;
    Tipoveiculo = tipo;
    Estadoveiculo = estado;
    this.custo = custo;
}

#endregion
```

E também o Override do ToString que nos permite mostrar os detalhes dos Veículos na consola:

```
#region Overrides

/// <summary>
/// Overrides da classe Veiculo
/// Permitem mostrar as informacoes dos veiculos na consola
/// </summary>

1 referência
public override string ToString()  //permite mostrar na consola as informacoes dos veiculos
{
    string outStr = String.Format("Id: {0}\t Tipo de Veiculo: {1}\t Estado: {2}\t Custo: {3}\n", id, Tipoveiculo, Estadoveiculo, Custo);
    return outStr;
}

#endregion
```

Foi também criada uma classe herança da classe Veículo, chamada "VeiculoEletrico" para a criação dos veículos elétricos do nosso programa:

VeiculoEletrico:

Esta classe irá ter os seus atributos próprios "autonomia" e "carga" e também foi definido um enumerado para armazenar o tipo de veículo elétrico (Bicicleta Elétrica ou Trotinete Elétrica):

```
/// <summary>
/// Enumerados da classe VeiculoEletrico
/// </summary>

[Serializable]
6 referências
public enum TIPOVEICULOELET    //enumerado dos tipos de veiculo
{
    BicicletaElet,
    TrotineteElet
}

/// <summary>
/// Classe VeiculoEletrico (Herança)
/// irá armazenar informações dos Veiculos elétricos
/// </summary>

[Serializable]
4 referências
public class VeiculoEletrico : Veiculo
{
    #region ATRIBUTOS

    /// <summary>
    /// Atributos da classe VeiculoEletrico
    /// </summary>

    int autonomia;  //autonomia do veiculo
    int carga;   //carga eletrica do veiculo

    TIPOVEICULOELET tipoveiculoelet;   //tipo de veiculo eletrico

    #endregion
```

As propriedades da mesma:

```
#region PROPRIEDADES

/// <summary>
/// Propriedades dos atributos da classe VeiculoEletrico
/// Permitem aceder aos atributos privados atavés dos getters e setters (encapsulamento)
/// </summary>

3 referências
public int Autonomia
{
    get { return autonomia; }
    set { autonomia = value; }
}

0 referências
public int Carga
{
    get { return carga; }
    set { carga = value; }
}

3 referências
public TIPOVEICULOELET Tipoveiculoelet
{
    get { return tipoveiculoelet; }
    set { tipoveiculoelet = value; }
}

#endregion
```

Os seus construtores:

```
#region CONSTRUTORES

/// <summary>
/// Construtores da classe VeiculoEletrico
/// Permitem criar um veiculo eletrico
/// Um dos construtores recebe argumentos e o outro não recebe (Principio do Polimorfismo)
/// </summary>

0 referências
public VeiculoEletrico()
{
    Idstatic++;
    Id = Idstatic;
    Tipoveiculoelet = TIPOVEICULOELET.TrotineteElet;
    Estadoveiculo = ESTADOVEICULO.Disponivel;
    Autonomia = 200;
}

2 referências
public VeiculoEletrico(TIPOVEICULOELET tipo, ESTADOVEICULO estado, double custo, int auton)
{
    //Idstatic++;
    Id = Idstatic;
    Tipoveiculoelet = tipo;
    Estadoveiculo = estado;
    Autonomia = auton;
    Custo = custo;

}
```

E o Override do ToString para a mesma:

```
#region Overrides

/// <summary>
/// Overrides da classe VeiculoEletrico
/// Permitem mostrar as informacoes dos veiculos eletricos na consola
/// </summary>

1 referência
public override string ToString()  //permite mostrar na consola as informacoes dos veiculos eletricos
{
    string outStr = String.Format("Id: {0}\t Tipo de Veiculo: {1}\t Estado: {2}\t Custo: {3}\t Autonomia: {4}\n", Id, Tipoveiculoelet,
        Estadoveiculo, Custo, Autonomia);
    return outStr;
}
#endregion
```

Aluguer:

Foram criados os atributos da classe aluguer, os atributos “user” e “vehicle” irão armazenar o Utilizador e o Veiculo de cada aluguer:

```
public class Aluguer
{
    #region ATRIBUTOS

    int id = 0;                 //numero de identificacao do aluguer
    static int idalu = 0;       //id estatico dos alugueres
    static int totalug = 0;             //total de alugueres
    double custoalug = 0;         //custo do aluguer

    int tempoalug;

    DateTime datainicio;     //data de inicio do aluguer
    DateTime datafim;        //data de fim do aluguer


    Utilizador user;   //armazena o utilizador que alugou o veiculo

    Veiculo vehicle;   //armazena o veiculo alugado pelo utilizador

    #endregion
```

Juntamente com as suas respetivas propriedades:

```
#region PROPRIEDADES

/// <summary>
/// Propriedades dos atributos da classe Aluguer
/// Permitem aceder aos atributos privados através dos getters e setters (encapsulamento)
/// </summary>

5 referências
public int Id
{
    get { return id; }
    set { id = value; }
}

4 referências
public int Idalu
{
    get { return idalu; }
    set { idalu = value; }
}

2 referências
public int Totalug
{
    get { return totalug; }
    set { totalug = value; }
}

3 referências
public double Custoalug
{
    get { return custoalug; }
    set { custoalug = value; }
}
```

```
3 referências
public DateTime Datainicio
{
    get { return datainicio; }
    set { datainicio = value; }
}

4 referências
public DateTime Datafim
{
    get { return datafim; }
    set { datafim = value; }
}

5 referências
public int Tempoalug
{
    get { return tempoalug; }
    set { tempoalug = value; }
}

5 referências
public Veiculo Vehicle
{
    get { return vehicle; }
    set { vehicle = value; }
}

5 referências
public Utilizador User
{
    get { return user; }
    set { user = value; }
}

#endregion
```

E os construtores desta classe:

```
#region CONSTRUTORES

/// <summary>
/// Construtores da classe Aluguer
/// Permitem criar um Aluguer
/// Um dos construtores recebe argumentos e o outro não recebe (Principio do Polimorfismo)
/// </summary>

1 referência
public Aluguer()
{
    Idalu++;
    Id = Idalu;
    Totalug++;
    Datainicio = DateTime.Now;
    Datafim = DateTime.Now.AddHours(1);
    Tempoalug = 0;
}

0 referências
public Aluguer(DateTime now, DateTime then, int tempo)
{
    Idalu++;
    Id = Idalu;
    Totalug++;
    Datainicio = now;
    Datafim = then;

    Tempoalug = tempo;
}

#endregion
```

E também o override que permite mostrar as informações de cada aluguer na consola:

```
#region Overrides

/// <summary>
/// Overrides da classe Aluguer
/// Permitem mostrar as informacoes dos alugueres na consola
/// </summary>

0 referências
public override string ToString()    //permite mostrar na consola as informacoes dos alugueres
{
    Console.WriteLine("\nInfo do Aluguer: ");
    string outStr = String.Format("Id Aluguer: {0}\t Data Inicio: {1}\t Data Fim: {2}\t Tempo do Aluguer: {3}\t Custo do Aluguer: {4}\n" +
        "\nUtilizador que Alugou: \nUtilizador: {5}\nVeiculo Alugado: \nVeiculo: {6}\n", Id, datainicio, datafim, Tempoalug, Custoalug, User, Vehicle);
    return outStr;
    Console.WriteLine("------------");
}

#endregion
```

Bicibox:

A classe Bicibox foi criada como classe estática e contém os seguintes atributos e propriedades:

```
public static class Bicibox
{
    #region ATRIBUTOS

    /// <summary>
    /// Atributos da classe Bicibox
    /// </summary>

    static int totveiculos;   //numero total de veiculos dentro da bicibox

    static int totveiculosout;   //numero total de veiculos fora da bicibox

    #endregion

    #region PROPRIEDADES

    /// <summary>
    /// Propriedades dos atributos da classe Bicibox
    /// Permitem aceder aos atributos privados atavés dos getters e setters (encapsulamento)
    /// </summary>

    3 referências
    static public int Totveiculos
    {
        get { return totveiculos; }
        set { totveiculos = value; }
    }

    4 referências
    static public int Totveiculosout
    {
        get { return totveiculosout; }
        set { totveiculosout = value; }
    }

    #endregion
}
```

DiretorIPCA:

Por último foi criada a classe estática DiretorIPCA que contém o seguinte atributo e a sua propriedade:

```
1 referência
static public class DiretorIPCA
{

    #region ATRIBUTOS

    /// <summary>
    /// Atributos da classe DiretorIPCA
    /// </summary>

    static string nome;  //nome do diretor

    #endregion


    #region PROPERTIES

    /// <summary>
    /// Propriedades dos atributos da classe DiretorIPCA
    /// Permitem aceder aos atributos privados atavés dos getters e setters (encapsulamento)
    /// </summary>

    1 referência
    static public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }

    #endregion

}
```

5. Desenvolvimento das Estruturas de dados e dos Métodos das classes (Fase 2):

DIVISÃO DO PROJETO EM CAMADAS:

Para facilitar a construção das estruturas de dados do projeto, este foi dividido em 4 camadas distintas:

BO (Business Objects):

Esta camada vai conter os atributos, propriedades e construtores das classes;

DL (Data Layer):

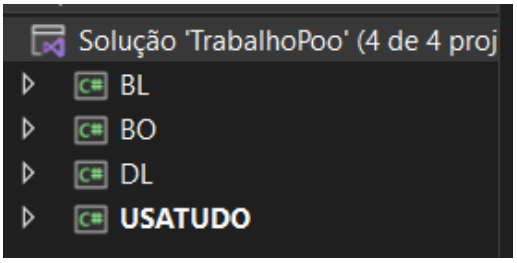
Esta é a camada que será responsável por armazenar os dados mais complexos das classes, estruturas de dados, listas, arrays e métodos;

BL (Business Layer):

Esta camada será responsável por chamar os métodos da camada anterior (DL) para estes poderem ser referidos no main da aplicação;

Usa Tudo:

Esta camada irá conter o Main da aplicação;



CONSTRUÇÃO DAS ESTRUTURAS DE DADOS E DOS MÉTODOS:

As estruturas de dados e métodos das classes estarão presentes na camada DL (Data Layer), assim teremos de criar classes especificas para guardar estas informações para cada classe:

DadosUtilizador:

Esta classe contém os métodos da classe Utilizador:

Método para consultar o Saldo do Utilizador:

```
1 referência
static public void ConsultaSaldo(Utilizador u)   //permite ao utilizador consultar o seu saldo
{
    Console.WriteLine("\nO saldo de " + u.Nome + " é: " + u.Saldo + " Euros\n");
}
```

Este método recebe um Utilizador e escreve na consola o seu saldo.

Método para Adicionar Saldo do Utilizador:

```
1 referência
static public void AdicionaSaldo(Utilizador u, int s)    //permite ao utilizador adcionar dinheiro ao seu saldo
{
    u.Saldo = u.Saldo + s;

    Console.WriteLine("\nForam Adicionados " + s + " euros à conta de " + u.Nome);

    Console.WriteLine("\nO saldo de " + u.Nome + " é: " + u.Saldo + " Euros\n");
}
```

Este método recebe um utilizador e uma quantia para ser adicionada ao saldo, adiciona essa quantia e mostra na consola o novo saldo.

Método mostrar o Histórico de Alugueres do Utilizador:

```
1 referência
static public void HistoricoAlugueresUtilizador(Utilizador u)   //Lista o historico de alugueres do utilizador
{
    Console.WriteLine("\nHISTORICO DE ALUGUERES DO UTILIZADOR:");

    foreach (Aluguer item in DadosBicibox.Historicobici)
    {
        if (item.User == u)
        {
            Console.WriteLine(item);
        }
    }
}
```

Este método recebe um utilizador e mostra o seu histórico de Alugueres.

Método mostrar o Veiculo que o Utilizador tem em posse:

```
1 referência
static public void MostraVeiculoUtilizador(Utilizador u)   //Mostra qual o veiculo que o utilizador tem em posse
{
    Console.WriteLine("\nVEIUCLO DO UTILIZADOR:");

    foreach (Aluguer item in DadosBicibox.Alugueresativos)
    {
        if (item.User == u)
        {
            Console.WriteLine("\nO Utilizador: \n" + u + "\n Tem em posse o veiculo: \n" +  item.Vehicle);
        }

    }
}
```

Este método recebe um utilizador e mostra o veiculo que o utilizador tem em posse.

Método que permite ao Utilizador alugar um Veiculo:

```
1 referência
static public void AlugaVeiculo(Veiculo v, Utilizador u, int tempo)   //permite ao utilizador alugar um veiculo
{
    DadosAluguer.AluguerVeiculo(v, u, tempo);
}

#endregion
```

Este método recebe um utilizador, um Veiculo e valor de tempo de aluguer e de seguida chama o método “AluguerVeiculo” contido em “DadosAluguer” para executar o aluguer do Veiculo.

DadosFuncionário:

Método que permite ao Funcionário colocar o veiculo na Bicibox:

```
1 referência
static public void ColocaBicibox(Veiculo v, Funcionario f)  //coloca veiculo dentro da bicibox
{
    DadosBicibox.AdicionaVeiculo(v);

    f.Numerocol++;

    Console.WriteLine("Veiculo Colocado com Sucesso!");
}
```

Este método recebe um Veiculo e um Funcionário e Adiciona o Veiculo à lista de Veiculos dentro da bicibox, removendo da lista de veículos fora da bicibox, chamando o Método "AdicionaVeiculo" de "DadosBicibox" e aumenta o número de Colocações de veículos na bicibox do Funcionário.

Método que mostra o número de colocações de Veiculos na Bicibox pelo Funcionário:

```
1 referência
static public void MostraNumeroColocacoesFuncionario(Funcionario f)   //mostra o numero de colocacoes efetuadas pelo funcionario
{
    Console.WriteLine("\nO Numero de colocacoes feitas pelo Funcionario é: " + f.Numerocol);
}
```

Este método recebe um funcionário e mostra o número de colocações do mesmo.

DadosMecanico:

Método que permite ao Mecanico alterar o estado dos Veiculos:

```
1 referência
static public void AlteraEstadoVeiculo(int idv, Mecanico m)   //permite alterar o estado de um veiculo dentro da bicibox
{

    foreach (Veiculo item in DadosBicibox.Veiculosbiciin)
    {
        if (item.Id == idv)
        {
            if (item.Estadoveiculo == ESTADOVEICULO.Avariado)
            {
                item.Estadoveiculo = ESTADOVEICULO.Disponivel;

                Console.WriteLine("Estado do Veiculo Alterado com Sucesso!");
            }

        }
    }

    m.Numerorepar++;
}
```

Este método recebe um Mecânico e um "id" do veiculo que ele pretende alterar o estado para "Disponível", percorrendo a lista de veículos dentro da Bicibox e alterando o estado do Veiculo com o "id" pretendido.

Método que mostra o número de reparações efetuadas pelo mecânico:

```
1 referência
static public void MostraNumeroReparacoesMecanico(Mecanico m)   //mostra o numero de reparacoes efetuadas pelo mecanico
{
    Console.WriteLine("\nO Numero de colocacoes feitas pelo Funcionario é: " + m.Numerorepar);
}
```

Este método recebe um Mecânico e mostra o número de reparações efetuadas pelo mesmo.

DadosAluguer:

Esta classe contém os métodos da classe Aluguer:

Método que cria um Aluguer:

```
2 referências
static public void AluguerVeiculo(Veiculo v, Utilizador u, int tempo)   //metodo que cria um aluguer
{
    if (v.Estadoveiculo == ESTADOVEICULO.Disponivel)
    {
        if (u.Saldo > 0)
        {
            Aluguer a = new Aluguer();

            a.Tempoalug = tempo;

            a.Custoalug = v.Custo * a.Tempoalug;

            u.Saldo = u.Saldo - a.Custoalug;

            a.Datainicio = DateTime.Now;
            a.Datafim = DateTime.Now.AddMinutes(tempo);

            a.Vehicle = v;
            a.User = u;

            DadosBicibox.Historicobici.Add(a);

            DadosBicibox.Alugueresativos.Add(a);

            u.Veiculouser = v;

            DadosBicibox.RemoveVeiculo(v);

            v.Estadoveiculo = ESTADOVEICULO.Ocupado;
            Console.WriteLine("\nAluguer Efetuado com Sucesso!");

        }
        else
        {
            Console.WriteLine("\nFalha no Aluguer! Utilizador não tem saldo Suficiente!");
        }
    }
    else
    {
        Console.WriteLine("\nFalha no Aluguer! Veiculo não se encontra Disponivel!");
    }
```

Este método recebe um Utilizador, um Veiculo e um determinado tempo. Com isto irá criar um Aluguer armazenando o Veiculo e o Utilizador e calculando o custo total, verificando se o saldo do utilizador é suficiente para efetuar este aluguer e também se o veiculo se encontra disponível, caso não se verifiquem estas condições irá enviar uma mensagem de erro. Irá também transferir o veiculo da lista que contém os veículos dentro da bicibox para a lista que contém os veículos fora da mesma e irá adicionar o aluguer criado à lista de histórico de alugueres e à lista de alugueres ativos da bicibox.

Método que cancela um Aluguer:

```
2 referências
static public void CancelarAluguer(int ida)  //metodo que cancela um aluguer
{

    for (int i = 0; i<DadosBicibox.Alugueresativos.Count; i++)
    {
        if (DadosBicibox.Alugueresativos[i].Id == ida)
        {
            DadosBicibox.Alugueresativos.Remove(DadosBicibox.Alugueresativos[i]);

            DadosBicibox.Alugueresativos[i].User.Veiculouser = null;


            DadosBicibox.AdicionaVeiculo(DadosBicibox.Alugueresativos[i].Vehicle);

            DadosBicibox.Alugueresativos[i].Vehicle.Estadoveiculo = ESTADOVEICULO.Disponivel;

            Console.WriteLine("Aluguer Cancelado com Sucesso!");
        }

}
```

Este método irá receber um “id” do aluguer que se pretende cancelar e remove esse aluguer da lista de alugueres ativos, remove o veiculo da posse do utilizador e altera o estado do mesmo de “Ocupado” para “Disponível”.

Método que para um Aluguer:

```
1 referência
static public void PararAluguer(int ida)    //metodo que para um aluguer
{
    for (int i = 0; i < DadosBicibox.Alugueresativos.Count; i++)
    {
        if (DadosBicibox.Alugueresativos[i].Id == ida)
        {
            if (DadosBicibox.Alugueresativos[i].Datafim == DateTime.Now)
            {
                CancelarAluguer(ida);
            }
            Console.WriteLine("Aluguer Parado com Sucesso!");
        }
    }
}

#endregion
```

Este método recebe um “id” do Aluguer que se pretende parar e coloca uma data de fim no mesmo e também chama o método “CancelarAluguer” para executar o seu cancelamento.

DadosBicibox:

Método que adiciona Veiculo na Bicibox:

```
3 referências
static public void AdicionaVeiculo(Veiculo a)   //adiciona veiculo na bicibox
{
    if (veiculosbiciin.Contains(a))
    {
        Console.WriteLine("\nVeiculo " + a.Id + " já está na Bicibox\n");
    }
    else
    {
        veiculosbiciin.Add(a);
        Bicibox.Totveiculos++;

        if (veiculosbicioout.Contains(a))
        {
            veiculosbicioout.Remove(a);
            Bicibox.Totveiculosout--;
        }
        else { }

    }

}
```

Este método recebe um Veiculo e adiciona o mesmo na lista de veículos dentro da bicibox, removendo da lista fora da bicibox caso este se encontre lá.

Método que Remove Veiculo da Bicibox:

```
2 referências
static public void RemoveVeiculo(Veiculo a)     //remove veiculo da bicibox
{
    if (veiculosbiciin.Contains(a))
    {
        veiculosbiciin.Remove(a);
        Bicibox.Totveiculos--;
        veiculosbicioout.Add(a);
        Bicibox.Totveiculosout++;


    }
    else
    {
        if (veiculosbicioout.Contains(a))
        {
            Console.WriteLine("\nVeiculo " + a.Id + " não está na Bicibox\n");
        }
        else
        {
            veiculosbicioout.Add(a);
            Bicibox.Totveiculosout++;
        }

    }
}
```

Este método recebe um veiculo e remove o mesmo da lista de veículos dentro da bicibox, adicionando o veiculo à lista fora da bicibox caso este ainda não se encontre lá.

Método que Lista os Veiculos dentro da Bicibox:

```
2 referências
static public void ListaVeiculosDentro()    //lista todos os veiculos dentro da bicibox (independentemente do estado)
{
    Console.WriteLine("\nTODOS OS VEICULOS DENTRO DA BICIBOX: \n");

    foreach (Veiculo item in veiculosbiciin)
    {
        Console.WriteLine(item);

    }

    Console.WriteLine("\nNumero de Veiculos dentro da Bicibox: \n" + Bicibox.Totveiculos);
}
```

Este método lista na consola os veículos que se encontram dentro da Bicibox e mostra o numero total de veículos que se encontram na mesma.

Método que Lista os Veiculos fora da Bicibox:

```
2 referências
static public void ListaVeiculosFora()    //lista todos os veiculosfora da bicibox (independentemente do estado)
{
    Console.WriteLine("\nTODOS OS VEICULOS FORA DA BICIBOX: \n");

    foreach (Veiculo item in veiculosbiciout)
    {
        Console.WriteLine(item);

    }

    Console.WriteLine("\nNumero de Veiculos fora da Bicibox: \n" + Bicibox.Totveiculosout);
}
```

Este método lista na consola os veículos que se encontram fora da Bicibox e mostra o numero total de veículos que se encontram fora da mesma.

Método que mostra o Histórico de Alugueres da Bicibox:

```
2 referências
static public void HistoricoAlugueresBicibox()    //lista o historico de alugueres da bicibox
{
    Console.WriteLine("\nHISTORICO DE ALUGUERES DA BICIBOX");

    foreach (Aluguer item in historicobici)
    {
        Console.WriteLine(item);

    }
}
```

Este método lista na consola o histórico de alugueres da Bicibox.

Método que mostra a lista de Alugueres Ativos da Bicibox:

```
2 referências
static public void ListaAlugueresAtivos()    //lista os alugueres ativos
{
    Console.WriteLine("\nLISTA DE ALUGUERES ATIVOS DA BICIBOX");

    foreach (Aluguer item in alugueresativos)
    {
        Console.WriteLine(item);

    }
}
```

Este método lista na consola os Alugueres ativos da Bicibox.

Método que mostra os Veiculos que se encontram disponíveis dentro da Bicibox:

```
2 referências
static public void MostraVeiculosDisponiveis()    //mostra apenas os veiculos dentro que se encontram disponiveis
{
    Console.WriteLine("\nLISTA DE VEICULOS DISPONIVEIS NA BICIBOX\n");

    foreach (Veiculo item in Veiculosbiciin)
    {
        if (item.Estadoveiculo == ESTADOVEICULO.Disponivel)
        {
            Console.WriteLine(item);
        }
    }


}
```

Este método lista na consola apenas os Veiculos que se encontram disponíveis dentro da Bicibox.

Método que permite editar um Veiculo:

```
1 referência
static public void EditarVeiculo(int idv, TIPOVEICULO tiponovo, ESTADOVEICULO estadonovo, double custonovo)  //permite editar um veiculo
{
    foreach (Veiculo item1 in Veiculosbiciin)
    {
        if (item1.Id == idv)
        {
            item1.Tipoveiculo = tiponovo;
            item1.Estadoveiculo = estadonovo;
            item1.Custo = custonovo;

            Console.WriteLine("Veiculo Editado com Sucesso!");
        }
        else
        {
            foreach (Veiculo item2 in Veiculosbiciout)
            {
                if (item2.Id == idv)
                {
                    item2.Tipoveiculo = tiponovo;
                    item2.Estadoveiculo = estadonovo;
                    item2.Custo = custonovo;

                    Console.WriteLine("Veiculo Editado com Sucesso!");
                }
            }
        }

    }


}
```

Este método irá receber os parâmetros respetivos aos campos dos atributos dos Veiculos, assim como o “id” do veiculo que se pretende editar, e irá efetuar as alterações pretendidas.

DadosDiretorIPCA:

Método que permite retornar todas as listas de uma vez:

```
1 referência
public static void GetAll()   //Mostra as listas de veiculos dentro da bicibox, fora, alugueres ativos e historico de alugueres da bicibox
{
    DadosBicibox.ListaVeiculosDentro();
    Console.WriteLine();
    DadosBicibox.ListaVeiculosFora();
    Console.WriteLine();
    DadosBicibox.ListaAlugueresAtivos();
    Console.WriteLine();
    DadosBicibox.HistoricoAlugueresBicibox();
    Console.WriteLine();
}
```

Este método irá mostrar na consola as listas de: Veiculos dentro da bicibox, Fora da Bicibox, Alugueres Ativos e Histórico de Alugueres da Bicibox.

Método que permite alterar o Nome do Diretor:

```
1 referência
public static void AlteraNome(string novo)   //permite alterar o nome do DiretorIPCA
{
    DiretorIPCA.Nome = novo;

    Console.WriteLine("Nome do Diretor Alterado com Sucesso!");
}

#endregion
```

Este método recebe um "nome" em formato string e altera o "nome" do Diretor para este novo nome em formato string.

6. Conclusão:

O desenvolvimento deste projeto permitiu a aquisição de novas práticas no desenvolvimento de um projeto na linguagem de programação C#, assim como a consolidação de conhecimentos adquiridos nas aulas da cadeira de Programação Orientada a Objetos.

Este Trabalho foi também bastante útil para a desenvolver o conhecimento sobre o paradigma de programação orientado a objetos, que tem crescido muito nos últimos anos.

7. Bibliografia/Web-grafia:

https://stackoverflow.com/

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/

https://www.w3schools.com/cs/index.php

https://www.tutorialsteacher.com/csharp

https://www.tutorialspoint.com/csharp/index.htm

https://www.javatpoint.com/c-sharp-tutorial