INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

RELATÓRIO DE TRABALHO PRÁTICO

Gestão de Urgência

PEDRO GOMES

ALUNO Nº 8626

TIAGO CRUZ

ALUNO Nº 9125

Trabalho realizado sob a orientação de:
Luís Ferreira

Linguagens de Programação II

Licenciatura em Engenharia de Sistemas Informáticos

Conteúdo

1 – Introdução

Neste relatório, o objetivo era a iniciação de um software para a gestão do bloco de Urgências de um hospital.

Na primeira parte realizamos a estrutura inicial das classes utilizadas e os respetivos métodos, tal como o diagrama de classes que sofreu alterações na segunda fase do trabalho.

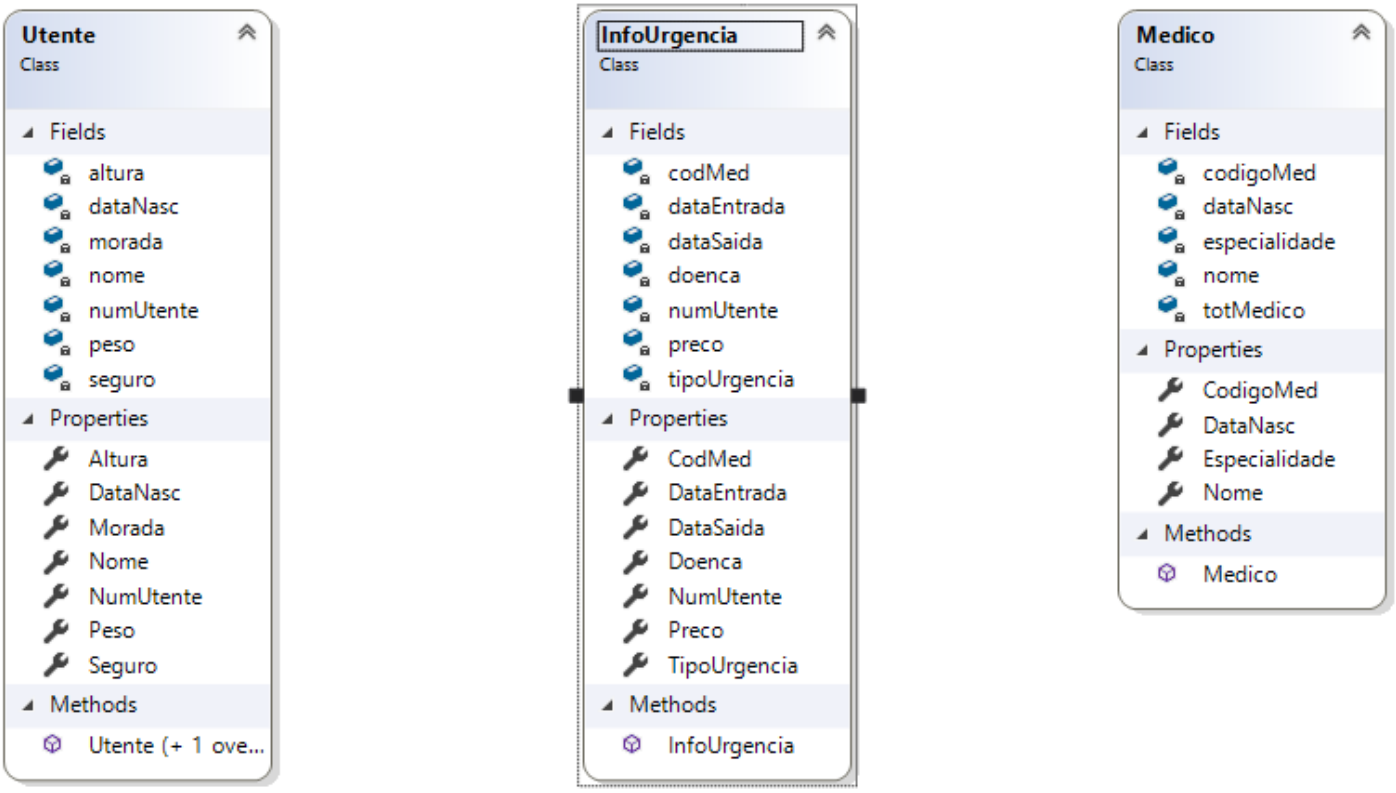
2 – Bibliotecas

Foi pedido pelo docente desta unidade curricular que este software fosse desenvolvido por camadas (bibliotecas) NTier, sendo elas a camada de Objetos, Dados, Regras de Negócio, Exceções e Gestão (main).

2.1 - Objetos

A camada dos dados é uma biblioteca onde são guardados todos os objetos utilizados pelos métodos. A biblioteca de Objetos comunica com todas as outras bibliotecas pois é, maioritariamente, a ligação principal entre bibliotecas, apesar de algumas bibliotecas comunicarem diretamente com outras. Isto assim acontece, pois, todas as bibliotecas necessitam de utilizar os objetos criados nesta camada.

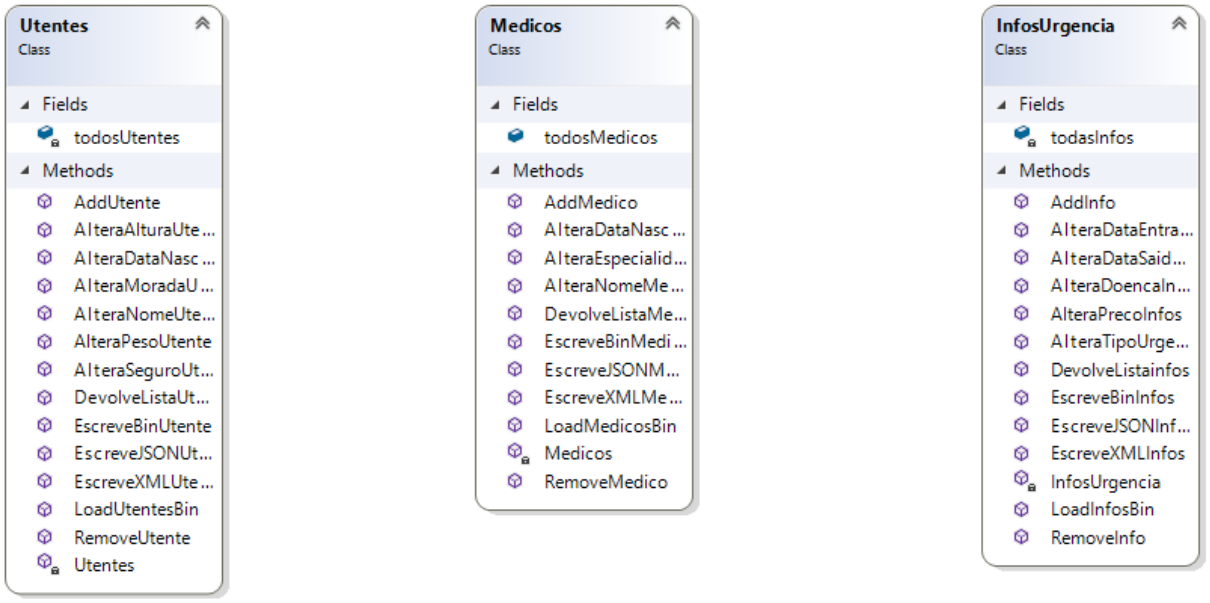
As classes definidas nesta camada foram a classe “Utente” que define as propriedades de um utente registado no hospital, “Medico” que define as propriedades de um médico regista e “InfoUrgencia” que define as propriedades de um conjunto de informações sobre a entrada de um utente nas urgências. Através do seguinte diagrama de classes podemos verificar como as classes anteriormente referidas estão estruturadas:



Originalmente apenas existiam duas classes de objeto (Utente e Médico), mas a pedido do docente da unidade curricular foi gerada uma terceira classe (InfoUrgencia) que define as variáveis a registar na entrada de um utente nas urgências, pois essas não definem o utente, mas são importantes para o tema do projeto.

2.2 – Dados

A camada dos Dados é uma biblioteca onde são definidos todos os métodos que utilizam e manipulam os objetos pelos métodos. A biblioteca de Dados comunica com a camada de Objetos e com a camada de Exceções. A camada contém 3 classes diferentes: "Utentes", que possui os métodos que manipulam a classe "Utente", "Medicos", que possui os métodos que manipulam a classe "Medico" e a classe "InfosUrgencia", que possui os métodos que manipulam a classe "InfoUrgencia". Através do seguinte diagrama de classes podemos ter uma noção quais são os métodos utilizados.



Como podemos observar todas as classes contém o mesmo tipo de métodos que irão ser descritas nos capítulos seguintes:

2.1.1 – Adicionar

Este método foi criado para adicionar um objeto a uma lista de objetos:

```
#region Adiciona utente
/// <summary>
/// Adiciona um utente à lista
/// </summary>
/// <param name="u">Utente</param>
/// <returns>true (caso seja adicionado à lista) / false (caso já exista na lista)</returns>
1 reference | fcctiago, 5 days ago | 1 author, 1 change
public static bool AddUtente(Utente u)
{
    //tenta executar o seguinte código
    try
    {
        //se a lista já contém o utente inserido
        if (todosUtentes.Contains(u))
        {
            return false;
        }
        //caso a lista não contenha
        todosUtentes.Add(u); //adiciona o utente u à lista
    }
    //caso haja um erro de inserção envia uma mensagem através da exception e
    catch (InsereException e)
    {
        throw e;
    }
    return true;
}
#endregion
```

Através do código apresentado podemos observar que o método recebe um objeto e tenta executar o seguinte: caso a lista de utentes contenha o objeto inserido, devolve um "false", caso não contenha, adiciona o objeto à lista. Caso haja um erro de inserção, o utilizador é avisado que inseriu mal os dados.

2.2.2 – Remover

Este método foi criado para remover um objeto de uma lista de objetos:

```
#region Remove utente
/// <summary>
/// Remove um utente da lista
/// </summary>
/// <param name="u">Utente</param>
/// <returns>true (caso consiga remover com sucesso)/ false (caso não encontre o utente na lista)</returns>
1 reference | fcctiago, 5 days ago | 1 author, 1 change
public static bool RemoveUtente(Utente u)
{
    //tenta executar o seguinte código
    try
    {
        //se a lista já contém o utente inserido
        if (todosUtentes.Contains(u))
        {
            todosUtentes.Remove(u); //remove o utente da lista
        }
        else //caso a lista não contenha o utente inserido
        {
            return false;
        }
        return true;
    }
    catch (InsereException e)
    {
        throw e;
    }
}
#endregion
```

Através do código apresentado podemos observar que o método recebe um objeto e tenta executar o seguinte: caso a lista de utentes contenha o objeto inserido, adiciona o objeto à lista devolve um “false”, caso não contenha, devolve um “false”. Caso haja um erro de inserção, o utilizador é avisado que inseriu mal os dados.

2.2.3 – Alterar

Estes métodos foram criados para alterar uma propriedade de um objeto de uma lista de objetos. Apenas iremos mostrar um dos métodos de alteração pois todos funcionam de maneira semelhante:

```
#region Alterar Nome
/// <summary>
/// Altera o nome do utente cujo número de utente é inserido
/// </summary>
/// <param name="nUtente">numero de utente desejado para alteração</param>
/// <param name="nome">novo nome</param>
/// <returns>true (se for efetuada a alteração)/false (se não for efetuada a alteração)</returns>
1 reference | 0 changes | 0 authors, 0 changes
public static bool AlteraNomeUtente(int nUtente, string nome)
{
    try
    {
        foreach (Utente u in todosUtentes)
        {
            if (u.NumUtente == nUtente)
            {
                u.Nome = nome; //iguala o nome do utente para a string de entrada
            }
            else return false;
        }
        return true;
    }
    catch (InsereException e)
    {
        throw e;
    }
}
#endregion
```

Através do código apresentado podemos observar que o método recebe o número de utente, que é único para cada utente e o valor que deseja alterar e tenta executar o seguinte: para cada utente na lista de utentes verifica se o número de utente existe. Caso exista, altera o nome, igualando o nome do utente à variável de entrada. Caso não contenha, devolve um “false”. Caso haja um erro de inserção, o utilizador é avisado que inseriu mal os dados.

2.2.4 – Devolver Lista de Objetos

Este método foi criado para criar uma cópia da lista de objetos para que esta possa ser utilizada fora da camada de dados.

```
#region Devolve lista
/// <summary>
/// Devolve a lista de utentes para ser utilizada fora da camada de dados
/// </summary>
/// <returns>uma cópia da lista de utentes</returns>
1 reference | 0 changes | 0 authors, 0 changes
public static List<Utente> DevolveListaUtentes()
{
    List<Utente> copiautentes = todosUtentes;
    return copiautentes;
}
#endregion
```

Ao analisar o código, podemos observar que ele cria uma lista nova e iguala-a à lista de objetos original e devolve a cópia para a mesma poder ser utilizada fora da camada.

2.2.5 – Guardar Lista de Objetos num ficheiro binário

Este método foi criado para criar um ficheiro binário e guardar a lista de utentes dentro do mesmo.

```
#region Escreve utentes em binário
/// <summary>
/// Insere os utentes registados num ficheiro binário
/// </summary>
/// <returns>true</returns>
1 reference | 0 changes | 0 authors, 0 changes
public static bool EscreveBinUtente()
{
    try
    {
        FileStream fs = new FileStream("Utentes.bin", FileMode.Create, FileAccess.ReadWrite); //cria ficheiro binário
        BinaryFormatter bf = new BinaryFormatter(); //cria serializador
        bf.Serialize(fs, todosUtentes); //serializa a lista
        fs.Flush();
        fs.Close();
        fs.Dispose(); //solta todos os recursos utilizados pela stream
    }
    catch(Exception e)
    {
        throw e;
    }

    return true;
}
#endregion
```

Ao analisar o código podemos observar que é criado uma nova “FileStream” que cria o ficheiro binário e um formatador binário. Após esses procedimentos a informação contida na lista de objetos é serializada para o ficheiro binário. Após esse processo o escritor é limpo, fechado e são soltos todos o recursos utilizados pelo “stream”.

2.2.6 – Guardar Lista de Objetos num ficheiro XML

Este método foi criado para criar um ficheiro XML e guardar a lista de utentes dentro do mesmo.

```
public static bool EscreveXMLUtentes()
{
    XmlDocument xmlDoc = new XmlDocument(); //Cria um novo ficheiro XML
    XmlNode rootNode = xmlDoc.CreateElement("utentes"); //Cria um nodo Raiz com o nome "utentes"
    xmlDoc.AppendChild(rootNode); //fecha o nodo raiz
    foreach (Utente u in todosUtentes)
    {
        XmlNode utenteNode = xmlDoc.CreateElement("utente"); //cria um nodo filho de "utentes" com o nome "utente"
        XmlNode nutNode = xmlDoc.CreateElement("numeroutente");//cria um nodo filho de "utente" com o nome "numeroutente"
        nutNode.InnerText = u.NumUtente.ToString(); //adiciona a propriedade "NumUtente" como texto interior do nodo "numerountente"
        utenteNode.AppendChild(nutNode);//fecha o nodo "numeroutente"
        XmlNode nomeNode = xmlDoc.CreateElement("nome");
        nomeNode.InnerText = u.Nome;
        utenteNode.AppendChild(nomeNode);
        XmlNode dnacNode = xmlDoc.CreateElement("datanascimemnto");
        dnacNode.InnerText = u.DataNasc.ToString();
        utenteNode.AppendChild(dnacNode);
        XmlNode pesoNode = xmlDoc.CreateElement("peso");
        pesoNode.InnerText = u.Peso.ToString();
        utenteNode.AppendChild(pesoNode);
        XmlNode alturaNode = xmlDoc.CreateElement("altura");
        alturaNode.InnerText = u.Altura.ToString();
        utenteNode.AppendChild(alturaNode);
        XmlNode morNode = xmlDoc.CreateElement("morada");
        morNode.InnerText = u.Morada;
        utenteNode.AppendChild(morNode);
        XmlNode segNode = xmlDoc.CreateElement("seguro");
        if (u.Seguro == true) segNode.InnerText= "Sim";
        else if (u.Seguro == false) segNode.InnerText = "Não";
        utenteNode.AppendChild(segNode);
        rootNode.AppendChild(utenteNode);

    }
    xmlDoc.Save("Utentes.xml");
    return true;
}
```

Ao analisar o código podemos observar que é criado uma novo ficheiro XML e é criado um nodo pai “utentes”. Posteriormente, para cada objeto na lista, é criado um nodo filho de “utentes” de nome “utente”. Após esse processo, são inseridos os nodos filhos de “utente”, que são as propriedades da classe “Utente”. Dentro desses nodos são inseridos no texto interior os dados das propriedades. Após todos os processos, o ficheiro é guardado.

2.2.7 – Guardar Lista de Objetos num ficheiro JSON

Este método foi criado para criar um ficheiro JSON e guardar a lista de utentes dentro do mesmo.

```
public static bool EscreveJSONUtentes()
{
    TextWriter file = new StreamWriter("Utentes.json"); //inicia o gravador de texto e cria um fic
    string json = JsonConvert.SerializeObject(todosUtentes, Newtonsoft.Json.Formatting.Indented);
    file.Write(json); //escreve a string convertida
    Object aux = JsonConvert.DeserializeObject(json); //desserializa a string que foi convertida
    file.Flush(); //limpa todos os buffers e faz com que quaisquer dados armazenados em buffer sej
    file.Close(); //fecha o gravador e limpa todos os recursos do sistema associados ao gravador
    return true;
}
```

Ao analisar o código podemos observar que é criado uma escritor de texto que cria o ficheiro binário e uma string onde vão ser guardados os dados convertidos na serialização da lista para JSON. Após esses procedimento, a informação é desserializada e o ficheiro é guardado e o escritor é limpo e fechado.

2.2.8 – Carregar ficheiro binário

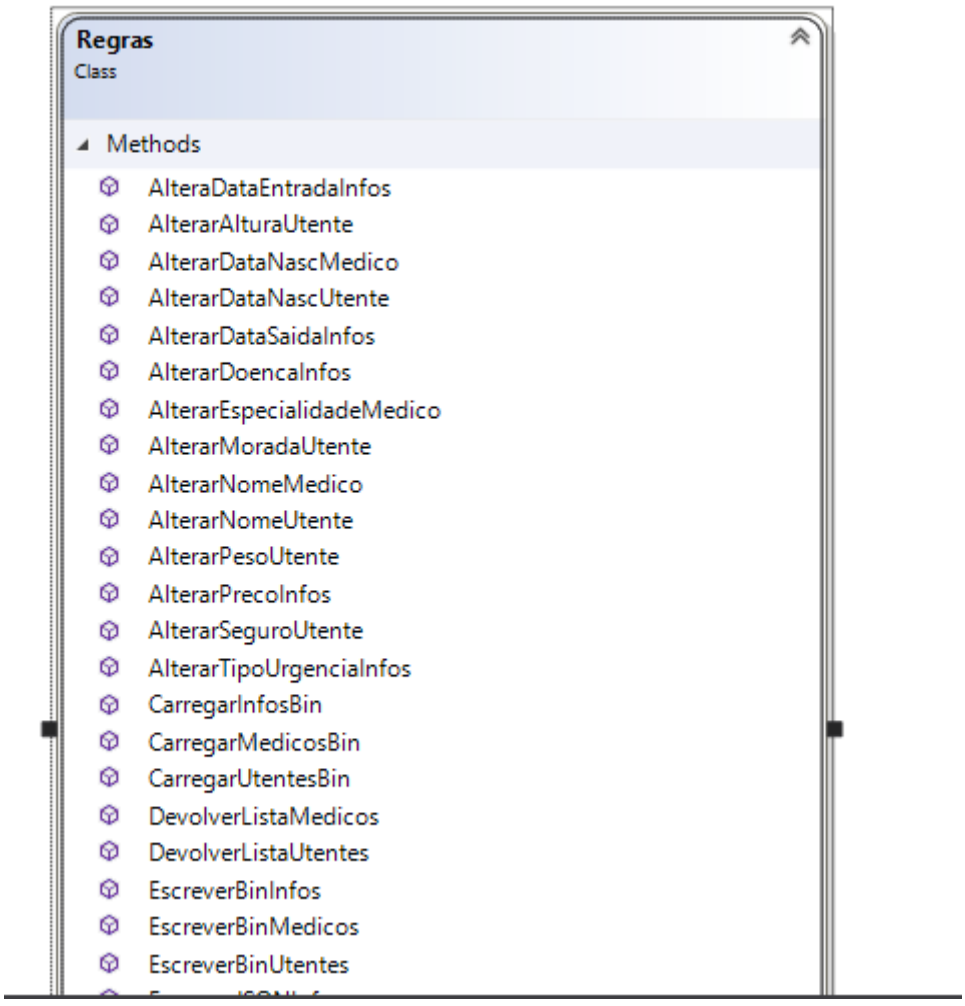
Este método foi criado para carregar um ficheiro binário e enviar os dados contidos no mesmo para a lista de objetos.

```
#region Carrega BIN utentes
/// <summary>
/// Carrega o ficheiro Bin
/// </summary>
/// <param name="fileName">Ficheiro a ser Carregado</param>
/// <returns></returns>
1 reference | 0 changes | 0 authors, 0 changes
public static bool LoadUtentesBin(string fileName)
{
    try
    {
        Stream s = File.Open(fileName, FileMode.Open, FileAccess.Read); //abre o ficheiro bin
        BinaryFormatter b = new BinaryFormatter(); //cria serializador
        todosUtentes = (List<Utente>)b.Deserialize(s); //deserializa a informação do ficheiro bin para a lista de utentes
        s.Flush();
        s.Close();
        s.Dispose();

        return true;
    }
    catch
    {
        throw new Exception("Erro");
    }
}

#endregion
```

Ao analisar o código podemos observar que é criado uma novo “Stream” que abre o ficheiro binário e é iniciado o formatador binário. Após esses procedimentos a informação contida no ficheiro é desserializada para a lista de objetos. Após esse processo o leitor é limpo, fechado e são soltos todos o recursos utilizados pelo “stream”.

2.3 – Regras

A camada das Regras é uma biblioteca onde são definidas as regras, ou seja, chama os métodos utilizados na camada de Dados para posteriormente serem utilizado na camada de Gestão. A biblioteca das Regras comunica com a camada de Dados e com a camada de Exceções. Através do seguinte diagrama de classes podemos ter uma noção quais são os métodos utilizados.



Ao analisar o diagrama podemos observar que são definidos os métodos a serem utilizados na camada da Gestão que chamam os métodos originais que manipulam os objetos.

2. 3 - Gestão

A camada de Gestão é uma biblioteca onde são utilizados os objetos e os métodos anteriormente definido nas camadas de Objetos e Regras. É também conhecida como a camada de interação, pois é a camada que interage com o utilizador da aplicação. A biblioteca de gestão comunica com a camada de Objetos e com a camada de Regras. Através do seguinte código podemos ter uma noção quais são os métodos e objetos utilizados.

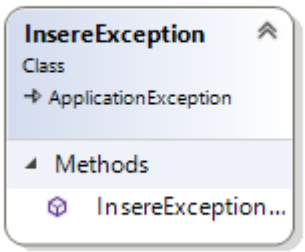
```
DateTime dnasc1, dnasc2;
List<Utente> listaUtentes = new List<Utente>();

DateTime.TryParse("31-03-1994 20:15", out dnasc1);
DateTime.TryParse("11-04-1956 10:30", out dnasc2);
Utente u1 = new Utente(1243435, "Tiago Filipe Carvalho da Cruz", dnasc1, 80.00, 1.71, "Rua Lino José Sousa Ferreira nº323", false);
Utente u2 = new Utente(1373439, "Rosa Maria Macedo Carvalho e Cruz", dnasc2, 62.00, 1.52, "Rua Lino José Sousa Ferreira nº323", true);
Regras.InserirUtente(u1);
Regras.InserirUtente(u2);
Regras.EscreverBinUtentes();
Regras.EscreverXMLUtente();
Regras.EscreverJSONUtente();
Regras.CarregarUtentesBin("Utentes.bin");
listaUtentes = Regras.DevolverListaUtentes();
foreach (Utente u in listaUtentes)
{
    Console.WriteLine("Nº de utente: " + u.NumUtente + "\n");
    Console.WriteLine("Nome: " + u.Nome + "\n");
    Console.WriteLine("Data de Nascimento: " + u.DataNasc + "\n");
    Console.WriteLine("Peso: " + u.Peso + "\n");
    Console.WriteLine("Altura: " + u.Altura + "\n");
    Console.WriteLine("Morada: " + u.Morada + "\n");
    if (u.Seguro == true) Console.WriteLine("Tem seguro?: Sim");
    else if (u.Seguro == false) Console.WriteLine("Tem seguro?: Não");
}
```

Ao analisar o código, podemos verificar que são criados 2 utentes e uma cópia da lista de utentes e adicionados à lista de utentes.

Posteriormente, são criados os ficheiros binário, XML e JSON. No fim desse processo, o ficheiro binário é carregado e são inseridos os dados do ficheiro na lista para no final mostrar esses mesmos dados inseridos.

2.4 – Exceções

A camada de Gestão é uma biblioteca onde são definidas todas as exceções, ou seja, todos os erros possíveis a ser feitos pelo utilizador serão avisados através de mensagens definidas nesta camada. A biblioteca de Exceções não comunica com nenhuma camada, mas as camadas de Dados e Objetos comunicam com ela. Através do seguinte diagrama de classes, podemos ter uma noção quais são os métodos utilizados.



Apenas adicionamos uma exceção de inserção pois não sabíamos que mais inserções poderíamos adicionar ao projeto.

3 - Conclusão

Após a conclusão deste projeto sentimos que deu para ter as noções essenciais da linguagem de programação C#, apesar de não terem sido implementada toda a matéria dada pelo docente desta unidade curricular.