



C#

Projeto Final

METODOLOGIA

- › Interprete o documento calmamente e com atenção.
- › Acompanhe a execução do exercício no seu computador.
- › Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- › Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- › Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

CONTEÚDO PROGRAMÁTICO

1. [Fase Inicial](#)
2. [Apresentação](#)
3. [Estrutura](#)
 - 3.1. [Classe Pergunta e Jogo](#)
 - 3.2. [Dificuldade e aleatório](#)
 - 3.3. [Classe Ajuda e derivadas](#)
 - 3.4. [Temporizador](#)
 - 3.5. [Prémios](#)
4. [Pontuações](#)
5. [Form Pontuação](#)
6. [Conteúdo](#)
 - 6.1. [Classe](#)
 - 6.2. [Gravar Pontuações](#)
 - 6.3. [Ler Pontuações](#)
 - 6.4. [Ordenação](#)

1. Fase Inicial

Neste módulo, iremos colocar em prática as matérias abordadas ao longo do curso, realizando um projeto final. O objetivo será criar um jogo, mais propriamente uma versão do conhecido **Quem quer ser Milionário**.

As regras do jogo são simples: temos 12 perguntas, cada pergunta tem 4 respostas em que só uma está correta. Vamos também ter 3 patamares de dificuldade, ou seja, 4 perguntas para cada patamar.

O jogo irá assentar sobre as seguintes regras:

- 30 segundos para responder a cada pergunta;
- O jogo é constituído por 12 perguntas;
- Quando acerta numa pergunta passa automaticamente para a próxima;
- Se acertar na última questão vence o jogo;
- Se errar em alguma pergunta perde;
- Existem 3 ajudas disponíveis;
- O jogador só pode utilizar cada ajuda uma vez durante o jogo todo;

As ajudas existentes no jogo são:

- **50:50** – Onde desativamos 2 respostas que estejam erradas, dando assim uma melhor hipótese ao jogador de acertar na resposta correta.
- **Audiência** – Pedimos ajuda à audiência, que nos dá a sua opinião sobre qual a resposta que consideram correta. Será implementado com um número aleatório.
- **Telefone** – Vamos ligar a um amigo ou familiar que, da mesma forma que a audiência, nos vai dar a sua opinião. Também será implementado com uma resposta aleatória.

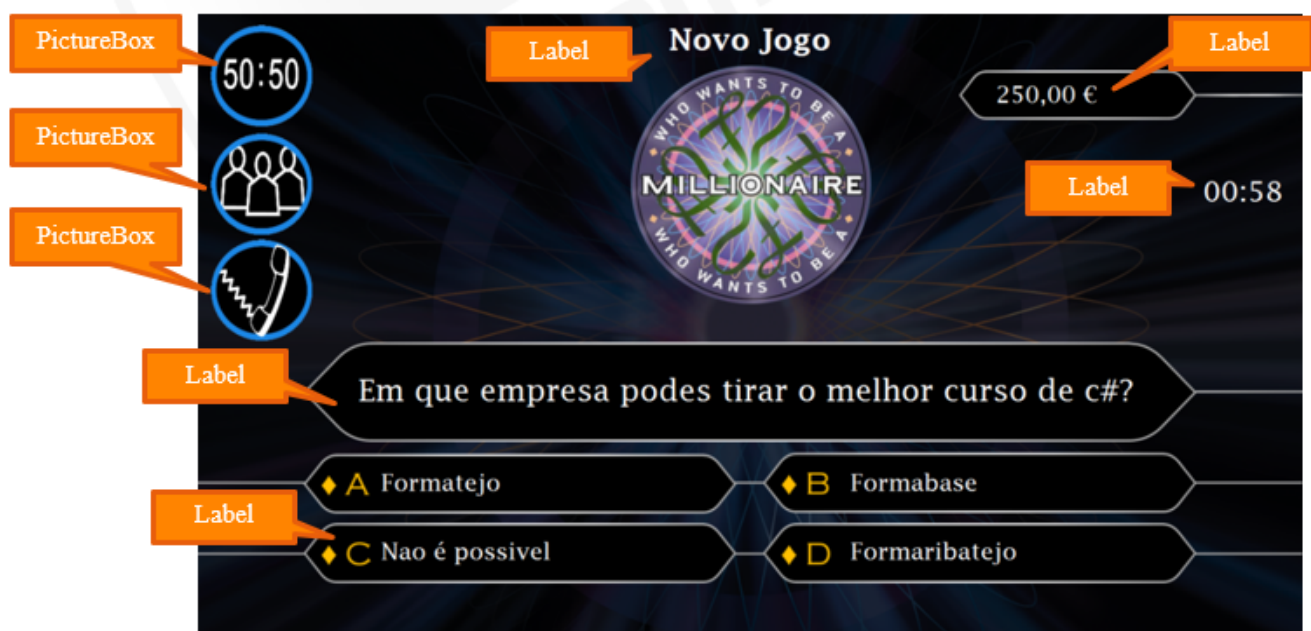
Os prémios serão mostrados por ordem crescente, sendo que, à medida que o jogador vai acertando nas perguntas, o prémio vai aumentando até chegar ao fim e ganhar o dinheiro!

A lista de prémios a ser considerada para o jogo deve ser a seguinte:

Pergunta	Prémio
1	50€
2	125€
3	250€
4	500€
5	750€
6	1250€
7	2500€
8	5000€
9	10 000€
10	50 000€
11	125 000€
12	250 000€

2. Apresentação

- › **Crie** um novo C# Project em **WindowsForms** com o nome `ProjetoFinalJogo`
- › No Form principal, **utilizando** os Objetos disponíveis para o Projeto, tente **recriar** a seguinte imagem:



› Pode **mudar** a **disposição** ao seu gosto

📄 os Objetos para este curso tem a imagem de fundo para o jogo, assim como as imagens para cada uma das ajudas.

📄 Para cada ajuda precisa para além de definir a propriedade `BackgroundImage` para colocar a imagem apropriada. Adicionalmente é necessário definir a propriedade `BackColor` como `Transparent`, de forma a não se ver o fundo branco da ajuda.

3. Estrutura

3.1. Classe Pergunta e Jogo

› **Crie** uma classe `Pergunta` contemplando os seguintes campos:

- Texto

- Resposta A, Resposta B, Resposta C, Resposta D
- Resposta Certa
- Dificuldade

📄 Todos os campos, à exceção da dificuldade, devem ser do tipo string.

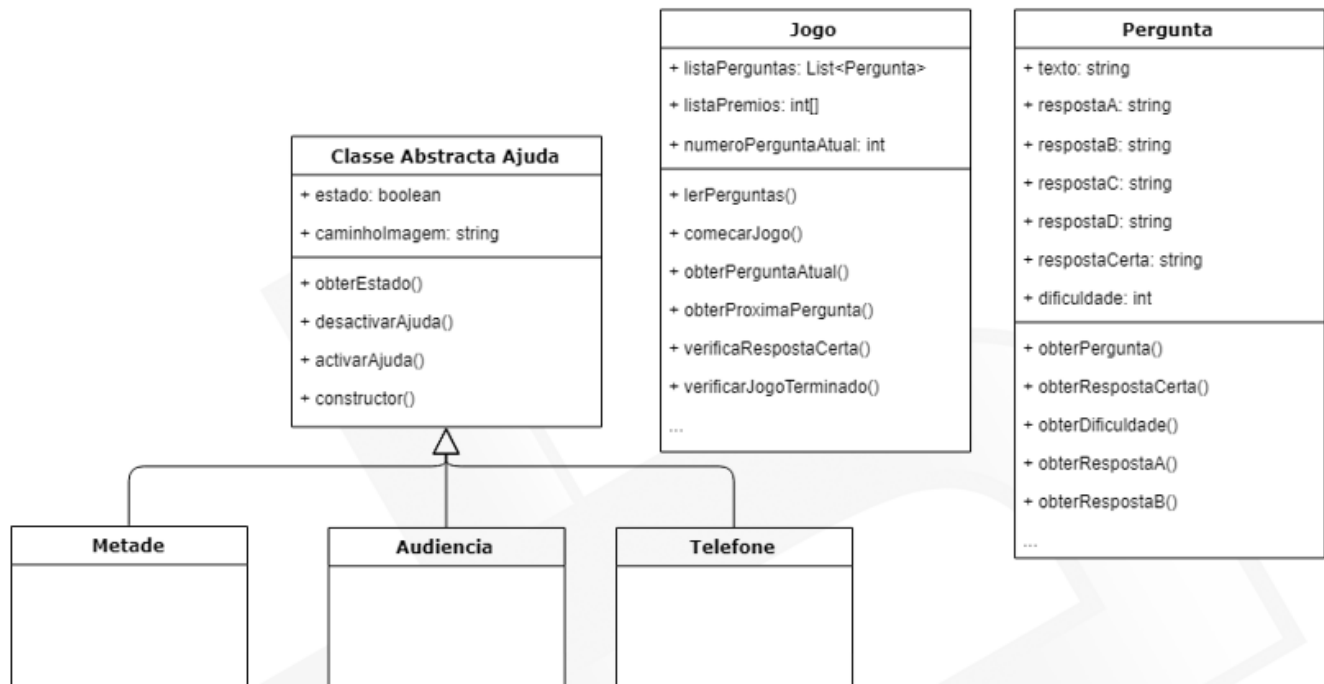
É necessário também criar métodos para obter os respetivos campos, seguindo assim os princípios de encapsulamento.

› **Crie** uma classe `Jogo`

Uma das boas práticas de programação consiste em separar a lógica do programa do aspeto visual. Assim, vamos criar esta classe `Jogo` que irá conter a lógica do jogo.

Sempre que seja necessário alterar o visual do jogo, como por exemplo, passar para a próxima pergunta, o mesmo deverá ser feito através do `form.cs`.

Para que o objetivo seja mais claro, o diagrama que se segue tem as classes e métodos a serem desenvolvidos. Este tipo de diagrama, no mundo da programação, chama-se **UML**, que significa *Unified Modeling Language* e segue determinadas regras consoante o que se pretende representar. Algumas das regras foram simplificadas de forma a torná-lo mais simples de perceber.



Antes de avançar, vamos ver mais detalhadamente o propósito de cada função:

Jogo:

lerPerguntas – Deverá ler todas as perguntas que se encontram no ficheiro `perguntas.csv` e guardá-las numa lista. Este ficheiro encontra-se na pasta **Objetos**. Esta leitura deve ser feita com base na leitura de um CSV como foi feita nos módulos anteriores.

obterPerguntaAtual – Devolver a pergunta atual.

obterProximaPergunta – Obtém a próxima pergunta a ser mostrada, caso ainda existam perguntas disponíveis.

verificarRespostaCerta – Recebe a resposta dada pelo jogador e verifica se a resposta é a correta.

jogoTerminado – Retorna verdadeiro caso o jogador tenha concluído as perguntas todas e falso caso ainda haja perguntas por responder ou o utilizador tenha errado a resposta.

As perguntas estão disponíveis num ficheiro `csv`. Deverá criar um método `lerPerguntas` que irá ler todas as perguntas e guardá-las numa lista.

A estrutura do ficheiro é a seguinte:

```
Pergunta;RespostaCerta;RespostaA;RespostaB;RespostaC;RespostaD;Dificuldade.
```

Um exemplo seria:

```
O Kitesurf é...;Um Desporto;Uma Ave;Uma Cidade;Um Desporto;Uma dança;1
```

Por agora ignore as ajudas e o temporizador e foque-se em conseguir colocar o **Form**, a classe `Jogo` e a classe `Pergunta` a funcionarem em conjunto.

Para que seja mais fácil guiar-se no que precisa de fazer, considere os seguintes pontos:

- Quando o Form do jogo abre, deve primeiramente ler as perguntas que existem no ficheiro através do método `lerPerguntas`
- A pergunta a ser mostrada no Form é obtida através do método `obterPerguntaAtual` da classe `Jogo`
- Quando o Form for iniciado, a pergunta atual deverá ser a primeira pergunta da lista
- Os vários campos da pergunta atual, `respostaA`, `respostaB`, `respostaC`, `respostaD`, `texto` devem ser atribuídos às respetivas labels no Form cada vez que avança para uma nova pergunta ou quando começa o jogo
- Cada vez que o utilizador clica numa resposta deve comparar com a resposta certa da pergunta atual para saber se avança para a próxima pergunta ou se o jogador perde o jogo

📄 A dificuldade deve ser ignorada neste ponto

› **Teste** as funcionalidades construídas até ao momento

📘 Neste ponto, o jogo deverá conseguir mostrar as perguntas e respetivas respostas e avançar nas perguntas à medida que o jogador acerta.

3.2. Dificuldade e aleatório

Para darmos mais realismo ao jogo, vamos implementar, não só as dificuldades, como tornar as perguntas aleatórias.

Quando o jogo começa, a dificuldade deve ser 1. Após o jogador acertar em 4 perguntas deverá passar à dificuldade 2, e assim sucessivamente, de 4 em 4 perguntas:

Pergunta	Dificuldade
1 - 4	1
5 - 8	2
9 - 12	3

Quando utilizar o método `obterProximaPergunta`, deve fazer uso da classe `Random` para obter uma pergunta aleatória.

› **Faça uso** do método `Next` da classe `Random` que recebe o valor máximo aleatório a gerar, e utilize o tamanho do array como valor máximo

Isto garante-lhe que, para um array de 10 perguntas, obtém um número entre 0 e 9, que corresponde às posições válidas do array.

📌 Tenha em mente que deve apenas obter uma pergunta que esteja na dificuldade em que o jogador se encontra.

📌 Se tiver alguma dúvida não hesite em pedir auxílio ao formador.

3.3. Classe Ajuda e derivadas

Crie uma **classe abstrata** `Ajuda`, e outras 3 classes derivadas de `Ajuda`, correspondentes às que existem no jogo:

- 50:50
- Telefone
- Audiência

i Note que uma classe não se poderá chamar 5050, pois o nome desta não pode começar com números. Neste caso, opte pelo nome **Metade** que foi sugerido no UML.

Estas classes vão derivar da classe `Ajuda` e vão herdar os seguintes métodos e variáveis:

- **estado** – Se a ajuda está ativa ou não num preciso momento.
- **caminholimagem** – De forma a facilitar a troca de imagens, quando a ajuda é desativada, indicamos o trajeto para a pasta que contém as mesmas.
- **obterEstado()** – Devolve o estado atual da ajuda.
- **obterCaminholimagem()** – Devolve o caminho atual para a imagem.
- **desativarEstado()** – Desativa a ajuda e altera a imagem, mostrando ao jogador que esta já não está disponível.
- **ativarAjuda()** – Método polimórfico diferente para cada subclasse, pois cada subclasse não tem a mesma função.
- **Construtor** – Será também necessário criar um construtor no sentido de facilitar a criação de cada ajuda.

No Form do jogo, cada vez que o utilizador clica numa imagem de Ajuda, tem de obter o objeto correspondente e chamar o método `ativarAjuda`. De seguida, deve atualizar a respetiva imagem no Form. A ajuda que o jogador utilizou terá de ficar indisponível até ao final do jogo.

No método `ativarAjuda` deve passar a pergunta em que se encontra e um array de booleanos que indique quais as ajudas a desativar, caso seja necessário desativar alguma. O retorno deste método deve ser o texto a apresentar ao jogador ou `null` caso não exista nenhum.


Desta forma a assinatura do método na classe `Ajuda` deve ser a seguinte:

```
public abstract string ativarAjuda(Pergunta pergunta, bool[] respostasD
```

Nas variantes de Ajuda do Público e Telefone, apenas deve retornar um texto que indique, de alguma forma, a escolha quer do público quer do telefone.

No caso da ajuda 50/50 precisa de alterar o array de respostas, desativando as respostas aleatórias e incorretas a serem retiradas.


Isto implica que, após chamar o método `ativarAjuda`, tem de consultar o texto devolvido para o mostrar numa `MessageBox` caso exista, e tem de consultar se houve alterações no array de booleanos para poder aceder às Labels do Form e retirar os respetivos textos.

 Não se esqueça que terá de utilizar os números aleatórios para as ajudas funcionarem da forma que pretendemos. Não dê demasiada relevância ao realismo das ajudas e foque-se mais na estrutura e incorporação das mesmas com o resto do jogo.

3.4. Temporizador

Para respeitar a regra dos 30 segundos por pergunta, é necessário utilizar **Timers**.

› **Adicione** um controlo **Timer** ao **Form** de Jogo e **defina** como **intervalo 1000**, que corresponde a um segundo

 Neste jogo é apenas necessário atualizar o cronómetro a cada segundo, por isso, definir 1000 na propriedade **Interval** do **Timer** é o mais apropriado.

› **Adicione** ao **Form** o evento **Tick** do **Timer**

É necessário atualizar o tempo que resta ao jogador e mostrá-lo na respetiva `Label`. O tempo deve ser mostrado com a seguinte formatação:

00:29

Se em alguma situação o tempo chegar a 0, o jogador perde de imediato. Tanto no cenário de vitória como derrota, deve ser apresentada uma mensagem ao utilizador através de uma `MessageBox` e o timer deve ser parado.

Sempre que o jogador avança numa mensagem, o tempo restante tem de voltar aos 30 segundos.

- ❏ Não avance para os próximos pontos sem garantir que tem as funcionalidades requisitadas até agora a funcionar corretamente.

3.5. Prémios

Para implementar os prémios é necessário criar um array de números inteiros.

- Após criar o array, **preencha** o mesmo com todos os valores dos prémios por **ordem crescente**

- ❏ Sempre que o jogador acertar na resposta certa, utilizamos o array para aceder ao próximo prémio.

```
int[] prémios = {  
    50,  
    125,  
    250,  
    500,  
    750,  
    1250,  
    2500,  
}
```

```
5000,  
10000,  
50000,  
125000,  
250000  
};
```

Quando começamos um novo jogo, o prémio é 0, pois o jogador ainda não acertou em nenhuma pergunta. À medida que o jogador acerta em perguntas vai subindo nos escalões de prémios, um por um.


4. Pontuações

Neste momento temos o jogo a funcionar, mas falta uma parte importante: as pontuações.

De forma a tornar o jogo mais interessante, vamos implementar o Top 4 das pontuações de todos os Jogadores. No entanto, para que isso possa acontecer, temos de remodelar o jogo.

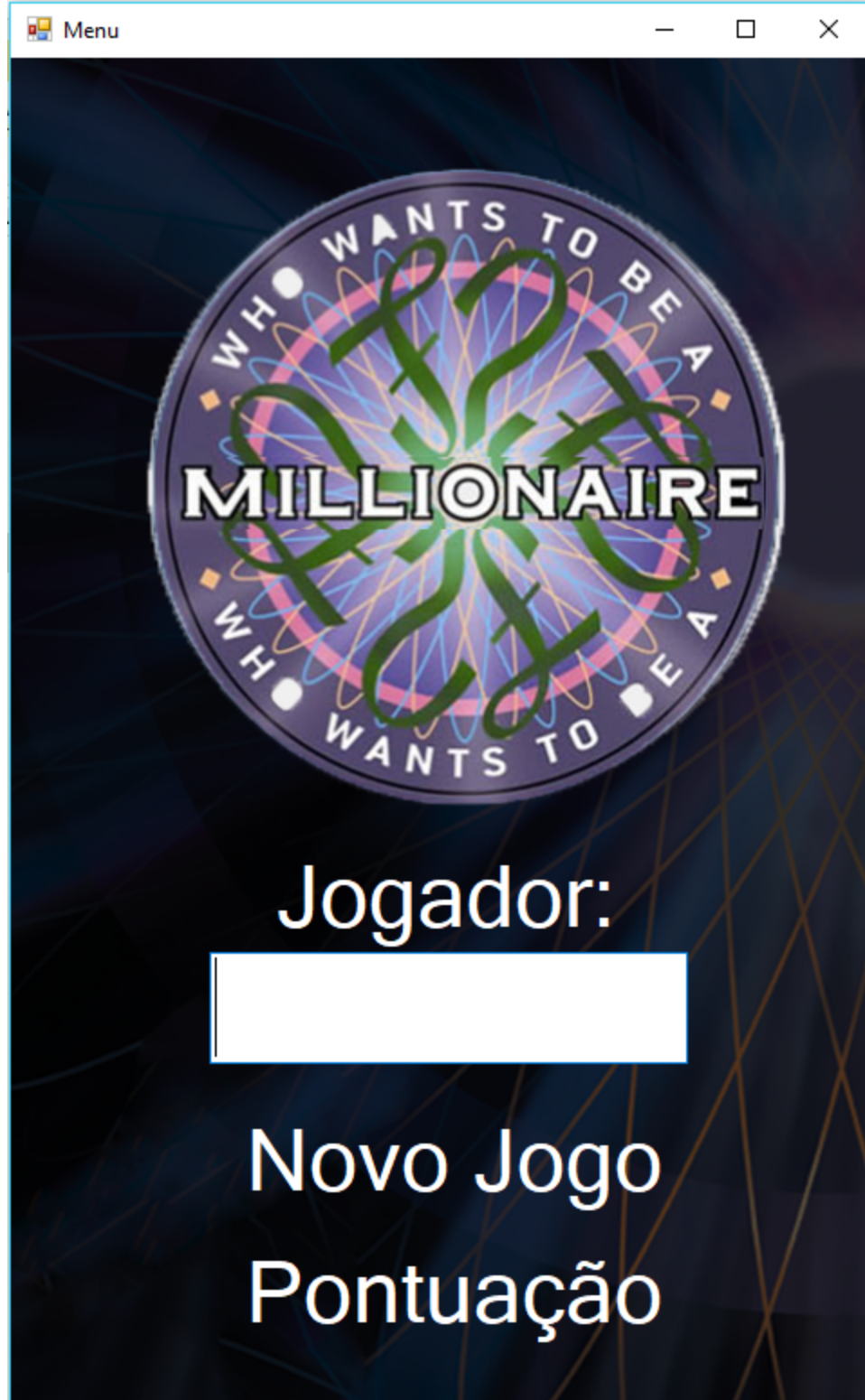
Começamos por criar um menu inicial, que irá surgir quando executar o jogo. Este menu terá duas opções:

- Iniciar um novo jogo
- Consultar as pontuações

 Ao iniciar um novo jogo cada utilizador coloca um nome e será a esse nome que as pontuações serão atribuídas.

› **Crie** um novo **Form** com o nome `MenuForm`

› **Utilizando** as **imagens** na pasta **objetos**, **ajuste** o **Form** ao seu gosto

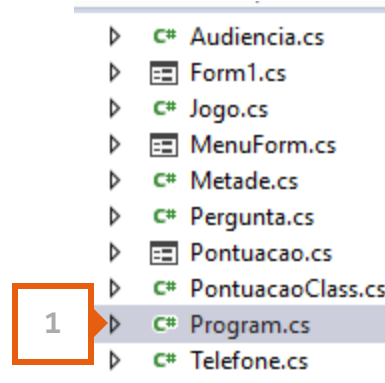


⚠ Se o utilizador tentar iniciar o jogo sem inserir o nome, terá de aparecer uma Mensagem de aviso.

Neste momento, o programa executa primeiro o **Form** do jogo, mas queremos alterar essa

funcionalidade para que ao executar o programa este abra primeiro o **Form Menu**.

Para isso, vamos alterar a classe **Program.cs** 1 .



› **Altere** o código realçado:

```
static class Program
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new MenuForm());
}
```

› **Confirme** que o novo Form é **executado** no início

📄 Se tiver alguma dúvida neste ponto, não hesite em pedir auxílio ao formador.

Sempre que for necessário abrir um novo Form, por exemplo, quando clicamos num botão com o intuito de passar para outra janela ou campo, utilizamos a seguinte sintaxe:

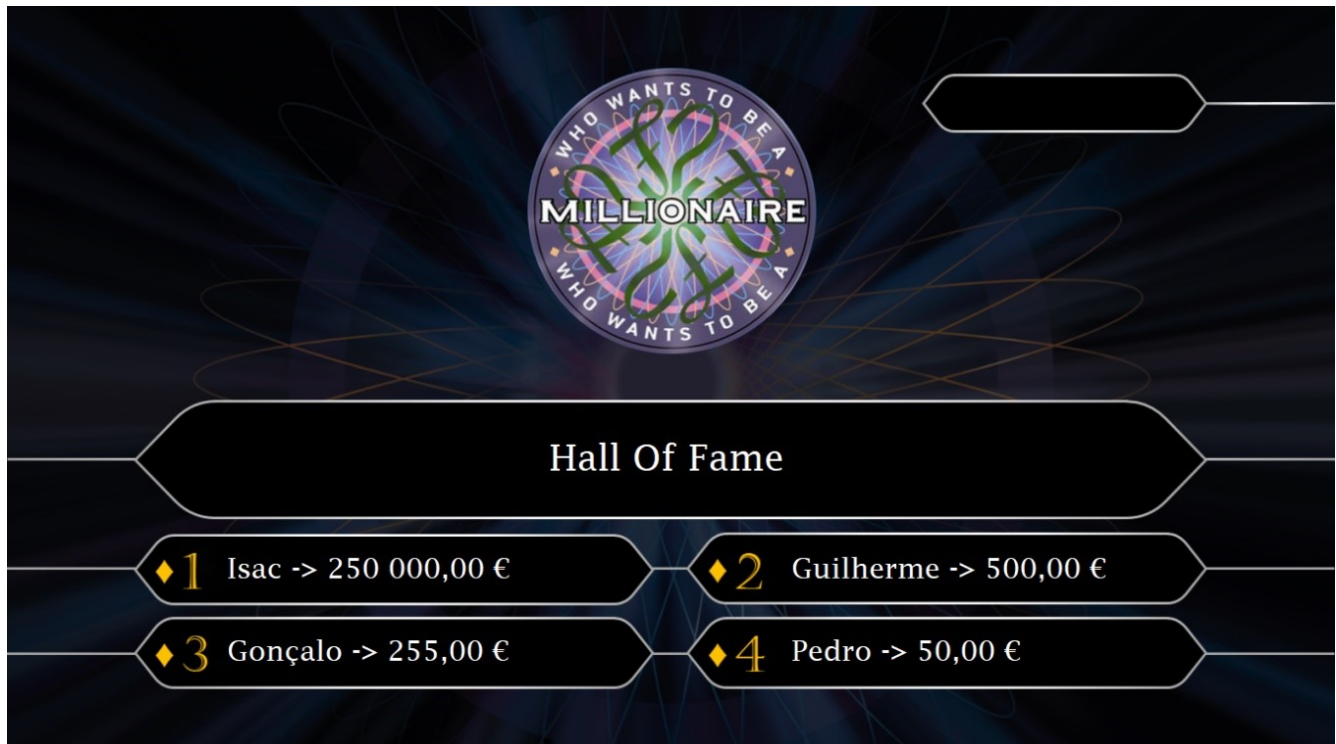
```
Form nomeForm = new Form(); //criar o outro Form
nomeForm.Show(); //abrir o Form criado
```

5. Form Pontuação

› Crie um novo **Form** com o nome `FormPontuacao`

📄 Este Form será utilizado para apresentar o Top 4 das pontuações, ordenadas do maior para o mais pequeno.

› Com as imagens disponíveis na pasta dos objetos, **recrie** a seguinte **imagem** para o novo **Form**:



6. Conteúdo

6.1. Classe

Para poder guardar o nome e a pontuação do jogador, criamos uma nova classe chamada `Pontuacao`.

Esta classe vai precisar de um campo para o valor do prémio e outro para o nome do jogador.

6.2. Gravar Pontuações

Em qualquer jogo as pontuações são mantidas mesmo tendo fechado o programa. Para que isso aconteça neste jogo, sempre que uma partida termine quer por vitória ou derrota, guardamos o nome e o prémio atingido pelo jogador.

› **Guarde** todas as **pontuações** num ficheiro `pontuacao.csv` com a seguinte estrutura:

```
Nome;Prémio
Joao;500000
Tiago;200000
```

Em quase todos os casos, é melhor ser feita a leitura de toda a informação quando o programa abre, normalmente numa `List`. Depois, quando o programa termina, voltamos a escrever no ficheiro, incorporando já todas as alterações feitas pelo utilizador.

Neste caso em particular, como apenas é necessário acrescentar uma nova linha no final do ficheiro, torna-se mais fácil abri-lo em modo **Append**, um modo que serve para acrescentar informação.

Para conseguir fazer esta escrita em modo **Append** utilize o seguinte código:

```
StreamWriter sw = File.AppendText("pontuacao.csv");
sw.WriteLine(nome + ";" + pontuacao);
sw.Close();
```

📄 Neste exemplo, o nome será o nome do jogador e a pontuação a quantia de dinheiro ganha pelo mesmo.

➤ **Aplique** este **código** no local onde o jogador vence ou perde o jogo, **ajustando** o nome e pontuação para os valores que tem no seu código

📄 O nome do jogador terá de ser passado do primeiro Form para o segundo Form. A forma mais simples e apropriada para fazer esta passagem é passar o valor pelo construtor do segundo Form.

6.3. Ler Pontuações

📄 Antes de avançar, confirme que tem pelo menos 4 pontuações gravadas no ficheiro.

Para a leitura das pontuações deve utilizar a classe `StreamReader` da mesma forma que foi utilizada em módulos anteriores. Idealmente irá guardar as pontuações numa lista:

```
List<Pontuacao> pontuacoes;
```

📄 Nesta linha de código, `Pontuacao` é a classe que representa um jogador e a sua respetiva pontuação.

Note que os elementos da lista são objetos da classe `Pontuacao`. Isto significa que à medida que lê as linhas do ficheiro de pontuações deve criar objetos desta classe `Pontuacao` e adicionar à lista.

Após lidas todas as pontuações, deve colocar manualmente as quatro primeiras nas respetivas labels do Form. O texto a ser apresentado deve ser composto pelo nome seguido de prémio e símbolo de euro. Algo similar a:

```
João -> 1250 €
```

- › **Teste** o código desenvolvido garantindo que consegue **ver** os quatro primeiros jogadores pontuados

i Não se deve ainda preocupar pelo facto de não aparecerem primeiro os jogadores com maior pontuação, pois isto será implementado no próximo ponto.

6.4. Ordenação

Depois de fazer a leitura das pontuações, terá de ordená-las para que sejam apenas mostradas as quatro pontuações mais altas. Para conseguir fazer isto, vamos utilizar um algoritmo de ordenação. O algoritmo que vamos utilizar é dos mais simples e mais conhecidos e chama-se **Bubble Sort**.

Para conseguirmos perceber como o **Bubble Sort** funciona, vamos ver um pequeno exemplo.

- › No Form `Pontuacao`, **crie** uma **função** com o nome `OrdenarArray` e **insira** o seguinte código:

```
public void ordenarArray()
{
    int[] array = {10, 1, 6, 7, 2, 3, 5};

    for(int i = 0; i < array.Length; i++)
    {
        for(int k = 1; k < array.Length - i; k++)
        {
            if(array[k - 1] < array[k])
            {
                int temp = array[k - 1];
                array[k - 1] = array[k];
                array[k] = temp;
            }
        }
    }
}
```

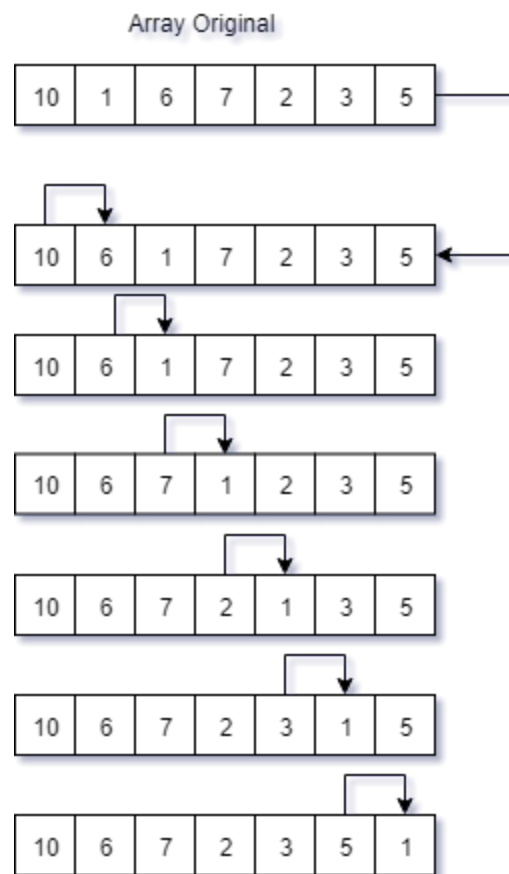
```
}  
}
```

📄 O código que inserimos é o código base do **Bubble Sort**.

O primeiro `for` vai definir a posição certa para apenas um elemento. O segundo `for` vai de casa em casa comparando com o número anterior do array e, se o anterior for menor que o da frente, efetua uma troca.

Isto faz com que na primeira passagem o menor valor fique no fim.

Veja o exemplo na ilustração seguinte:



Esta troca define a posição certa para o elemento menor. O primeiro `for` repete esse processo do início, mas considerando menos um elemento, o último, pois esse já ficou na posição certa. Nesta segunda passagem vai colocar o segundo menor na penúltima posição. O processo repete-se colocando o terceiro menor na ante penúltima casa, e assim sucessivamente até estar tudo ordenado.

O código responsável pela troca de valores entre as duas casas do array é o seguinte:

```
int temp = array[k - 1];  
array[k - 1] = array[k];  
array[k] = temp;
```

Para não perdermos a informação das variáveis, guardamos um dos valores numa variável temporária.

› **Insira** o seguinte código antes do final do método:

```
for(int i = 0 ; i < 3 ; i++)  
{  
    MessageBox.Show(array[i].ToString());  
}
```

📄 Este bloco de código irá mostrar as três primeiras posições do array.

› Faça o **teste**

📄 Irão aparecer os números 10, 7, 6.

Agora que já conhece o algoritmo de ordenação, adapte o código das pontuações do projeto para que apareçam apenas as quatro pontuações mais altas.

› **Confirme** que todo o projeto está a **funcionar**

📄 Se tiver alguma questão, peça auxílio ao formador.