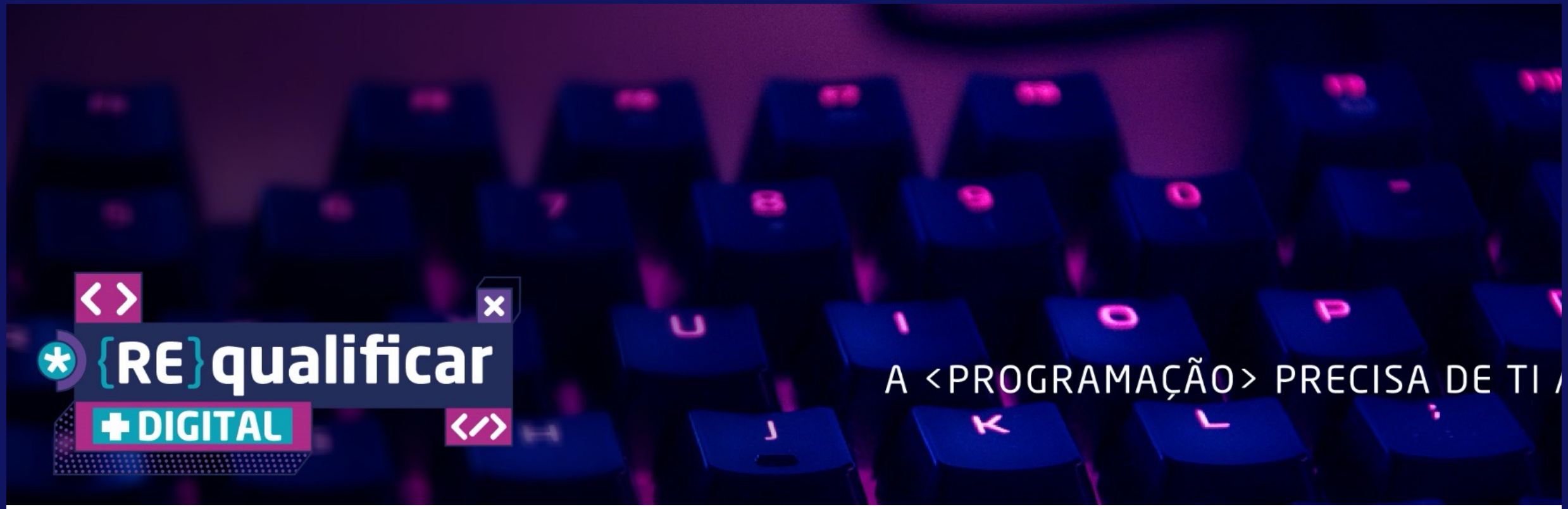


Laboratório Web

POO

UNIVERSIDADE da MADEIRA

Região Autónoma da Madeira
Governo Regional

startup madeira

Programação Orientada a Objetos

O que é POO?

- É um paradigma de programação, que usa objetos e suas interações para construir programas

O que é um objeto?

- Objetos de software modelam objetos do mundo real ou conceitos abstratos (que também são considerados objetos).
  - Exemplos de objetos do mundo real são pessoas, carros.
  - Exemplos de objetos abstratos são as estruturas de dados pilha, fila, lista e árvore.

Programação Orientada a Objetos

Nos objetos do mundo real, assim como nos objetos abstratos, podemos distinguir os seguintes dois grupos de suas características:

- **Estados** – são as características do objeto que o definem de uma maneira e o descrevem em geral ou em um momento específico
- **Comportamento** – estas são as ações distintivas específicas, que podem ser feitas pelo objeto
  - E.g., cão. Os estados do cão podem ser "nome", "cor da pele" e "raça", e seu comportamento - "latindo", "sentado" e "andando”

Objetos em POO combinam dados e os meios para seu processamento num só

- **Dados** – embutidos em variáveis de objetos, que descrevem seus estados.
- **Métodos** – para construir os objetos.

Classes

- A classe define características abstratas de objetos.
- Fornece uma estrutura para objetos ou um padrão que usamos para descrever a natureza de alguma coisa
- Estão inseparavelmente relacionadas aos objetos. Além disso, cada objeto é uma instância de exatamente uma classe específica.
- Classe e um objeto (instância da classe). E.g., uma classe Dog e um objeto Lassie, que é uma instância da classe Dog. A classe Dog descreve as características de todos os cães, enquanto Lassie é um determinado cão.
- As classes fornecem modularidade nos programas orientados a objetos. As suas características devem ser relevantes para um contexto comum de forma a serem compreendidas.

Classes, Atributos e Comportamento

- A classe define as características de um objeto (atributos) e seu comportamento (ações que podem ser realizadas pelo objeto).
- Os atributos da classe são definidos como próprias variáveis no corpo da classe.
- O comportamento dos objetos é modelado pela definição de métodos em classes.

- Algum exemplo?

Objetos – Instâncias de Classes

- Cada objeto é uma instância de apenas uma classe e é criado de acordo com um padrão dessa classe.
- A criação do objeto de uma classe definida é chamada de **instanciação (criação)**. A **instância é o próprio objeto**, que é criado em tempo de execução.

Exemplo de classe

```
public class Cat
{
  // Field name
  private string name;
  // Field color
  private string color;

  public string Name
  {
    // Getter of the property "Name"
    get
    {
      return this.name;
    }
    // Setter of the property "Name"
    set
    {
      this.name = value;
    }
  }

  public string Color
  {
    // Getter of the property "Color"
    get
    {
      return this.color;
    }
    // Setter of the property "Color"
    set
    {
      this.color = value;
    }
  }

  // Default constructor
  public Cat()
  {
    this.name = "Unnamed";
    this.color = "gray";
  }

  // Constructor with parameters
  public Cat(string name, string color)
  {
    this.name = name;
    this.color = color;
  }

  // Method SayMiau
  public void SayMiau()
  {
    Console.WriteLine("Cat {0} said: Miauuuuuu!", name);
  }
}
```

```
static void Main()
{
  Cat firstCat = new Cat();
  firstCat.Name = "Tony";
  firstCat.SayMiau();

  Cat secondCat = new Cat("Pepy", "red");
  secondCat.SayMiau();
  Console.WriteLine("Cat {0} is {1}.",
    secondCat.Name, secondCat.Color);
}
```

Criando e usando objetos

- A criação de objetos a partir de classes previamente definidas durante a execução do programa é realizada pelo operador **new**.

```
Cat someCat = new Cat();

Cat someCat = new Cat("Johnny", "brown");
```

Aceder aos campos de um objeto

- Para aceder aos campos e propriedades de um determinado objeto usam o operador. (ponto) colocado entre os nomes do objeto e o nome do campo (ou da propriedade).
- O operador . não é necessário caso queiramos aceder ao campo ou propriedade de determinada classe no corpo de um método da mesma classe.
- Podemos aceder aos campos e as propriedades para extrair dados ou atribuir novos dados.

Chamar Métodos de Objetos

- A chamada dos métodos de um determinado objeto é feita através do operador de invocação () e com a ajuda do operador . (ponto).
- O operador ponto não é obrigatório caso o método seja chamado no corpo de outro método da mesma classe.
- A chamada de um método é realizada pelo nome seguido de () ou (<parâmetros>) para o caso em que passamos alguns argumentos

```
class CatManipulating
{
   static void Main()
   {
      Cat myCat = new Cat();
      myCat.Name = "Alfred";

      Console.WriteLine("The name of my cat is {0}.",myCat.Name);
      myCat.SayMiau();
   }
}
```

Chamar Métodos de Objetos

- A chamada dos métodos de um determinado objeto é feita através do operador de invocação () e com a ajuda do operador . (ponto).
- O operador ponto não é obrigatório caso o método seja chamado no corpo de outro método da mesma classe.
- A chamada de um método é realizada pelo nome seguido de () ou (<parâmetros>) para o caso em que passamos alguns argumentos

```
class CatManipulating
{
   static void Main()
   {
      Cat myCat = new Cat();
      myCat.Name = "Alfred";

      Console.WriteLine("The name of my cat is {0}.",myCat.Name);
      myCat.SayMiau();
   }
}
```

Construtores

- O construtor é um método especial da classe, que é chamado automaticamente ao criar um objeto desta classe
- O construtor realiza a inicialização de seus dados (é o seu propósito).
- O construtor não possui tipo de valor retornado e o seu nome não é aleatório, obrigatoriamente coincide com o nome da classe.
- O construtor pode ser com ou sem parâmetros. Um construtor sem parâmetros também é chamado de construtor sem parâmetros.

Construtores

- O construtor é um método especial da classe, que é chamado automaticamente ao criar um objeto desta classe
- O construtor realiza a inicialização de seus dados (é o seu propósito).
- O construtor não possui tipo de valor retornado e o seu nome não é aleatório, obrigatoriamente coincide com o nome da classe.
- O construtor pode ser com ou sem parâmetros. Um construtor sem parâmetros também é chamado de construtor sem parâmetros - **parameterless constructor**

Construtor com parâmetros

- O construtor pode receber parâmetros, assim como qualquer outro método.
- Cada classe pode ter uma contagem diferente de construtores com uma única restrição – a contagem e o tipo de seus parâmetros devem ser diferentes (assinatura diferente).
- Ao criar um objeto desta classe, um dos construtores é chamado.
  - Na presença de vários construtores em uma classe o construtor apropriado é escolhido automaticamente pelo compilador de acordo com o conjunto de parâmetros fornecido ao criar o objeto. Usamos o princípio da melhor correspondência.

Construtor com parâmetros

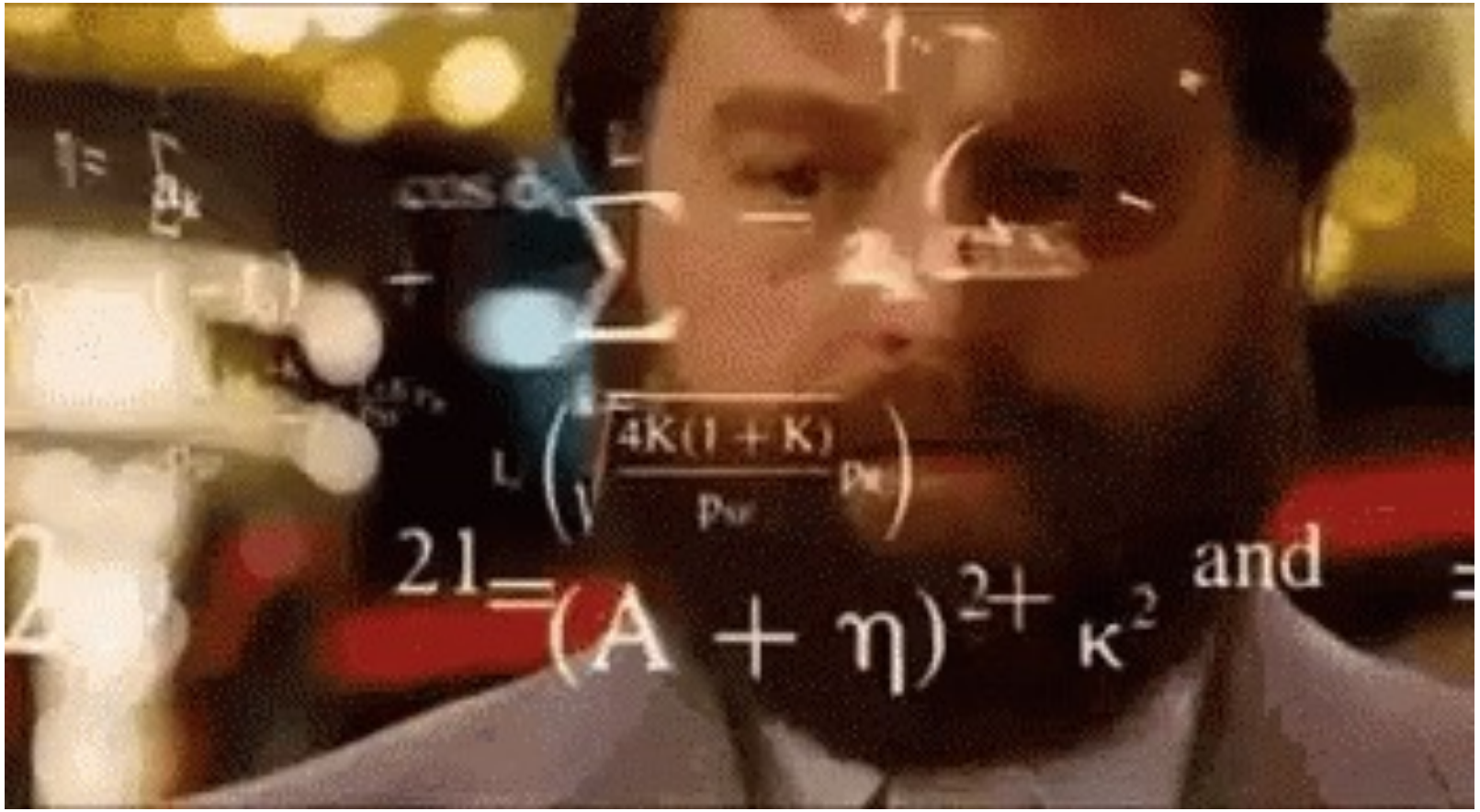
```
public class Cat
{
   // Field name
   private string name;
   // Field color
   private string color;

   …

   // Parameterless constructor
   public Cat()
   {
      this.name = "Unnamed";
      this.color = "gray";
   }

   // Constructor with parameters
   public Cat(string name, string color)
   {
      this.name = name;
      this.color = color;
   }

   …
}
```

```
class CatManipulating
{
   static void Main()
   {
      Cat someCat = new Cat();

      someCat.SayMiau();
      Console.WriteLine("The color of cat {0} is {1}.",


       someCat.Name, someCat.Color);

      Cat someCat = new Cat("Johnny", "brown");

      someCat.SayMiau();
      Console.WriteLine("The color of cat {0} is {1}.",
         someCat.Name, someCat.Color);
   }
}
```

Campos estáticos e métodos

- Os dados considerados até agora implementam estados dos objetos e estão diretamente relacionados a instâncias específicas das classes.
- Na POO existem métodos de categorias especiais, que estão associados ao tipo de dado (classe), e não à instância específica (objeto). São chamamos de **membros estáticos** porque são independentes de objetos concretos.
- **Quando usar campos e métodos estáticos?**
  - Vamos interpretar a classe como uma categoria de objetos e o objeto como representante dessa categoria.
  - Os **membros estáticos** refletem o estado e o comportamento **da própria categoria**, e os **não estáticos** o estado e o comportamento das representações separadas da categoria.

Campos estáticos e métodos



Classes estática

- Uma classe estática é basicamente o mesmo que uma classe não estática, mas há uma diferença: **uma classe estática não pode ser instanciada**.
- Como não há variável de instância, os membros de uma classe estática são chamados usando o próprio nome da classe.
- Uma classe estática pode ser usada como um container para conjuntos de métodos que operam apenas em parâmetros de entrada e não precisam obter ou definir nenhum campo de instância interno (e.g., classe Math, classe temperatura)

```
double dub = -3.14;
Console.WriteLine(Math.Abs(dub));
Console.WriteLine(Math.Floor(dub));
Console.WriteLine(Math.Round(Math.Abs(dub)));
```

Classes estáticas

Principais características de uma classe estática:

- Contém apenas membros estáticos.
- Não podem ser instanciadas.
- Estão seladas - não podem ser herdadas
- Não pode conter construtores de instância.

- Criar uma classe estática é basicamente o mesmo que criar uma classe que **contém apenas membros estáticos e um construtor privado**.
  - Um construtor privado impede que a classe seja instanciada.
- A vantagem de usar uma classe estática é que o compilador pode verificar se nenhum membro de instância foi adicionado acidentalmente. O compilador garante que instâncias dessa classe não possam ser criadas.

Métodos estáticos

- Uma classe não estática pode conter métodos, campos, propriedades ou eventos estáticos.
- O membro estático pode ser chamado numa classe mesmo quando nenhuma instância da classe foi criada. O membro estático é sempre chamado pelo nome da classe, não pelo nome da instância.
- Os métodos estáticos podem ser overloaded, mas não overwritten, porque pertencem à classe e não a qualquer instância da classe.
- É mais comum declarar uma classe não estática com alguns membros estáticos do que declarar uma classe inteira como estática.
  - Dois usos comuns de campos estáticos são manter uma contagem do número de objetos que foram instanciados ou armazenar um valor que deve ser partilhado entre as instâncias.
  - E.g., Pense nos Ids da UMa. Cada Aluno/Docente/Funcionário é um objeto separado e cada pessoa precisa do seu próprio ID exclusivo. Por isso devemos acompanhar o último ID atribuído. Esta seria a variável estática na classe.

Recall: classes e objetos

- A classe é a chamada definição (especificação) de um determinado tipo de objetos do mundo real.
- A classe representa um padrão, que descreve os diferentes estados e comportamentos de determinados objetos (as cópias), que são criados a partir dessa classe (padrão).
- Objeto é uma cópia criada a partir da definição (especificação) de uma determinada classe, também chamada de instância.
- Quando um objeto é criado pela descrição de uma classe dizemos que o objeto é do tipo "nome da classe".

Classes

Elementos da classe:

- Declaração–linha onde declaramos o nome da classe,
- Corpo de classe –definido logo após a declaração da classe, entre "{" e "}". O conteúdo dentro dos {} é conhecido como corpo da classe. Os elementos da classe, que estão numerados abaixo, fazem parte do corpo.
  - Construtor – é usado para criar novos objetos.
  - Campos – são variáveis, declaradas dentro da classe (conhecidas como variáveis-membro). Os valores, que estão nos campos, refletem o estado específico do objeto dado, mas apesar disso existem outros tipos de campos, chamados estáticos, que são compartilhados entre todos os objetos (e.g., id aluno).
  - Propriedades – esta é a maneira de descrever as características de uma determinada classe (e.g., get, set).
  - Métodos –blocos de código de programação. Realizam ações particulares e através deles os objetos atingem o seu comportamento. Os métodos executam a lógica de programação implementada (algoritmos) e o tratamento dos dados.

Classes

Elementos da classe:

- Declaração–linha onde declaramos o nome da classe,
- Corpo de classe –definido logo após a declaração da classe, entre "{" e "}". O conteúdo dentro dos {} é conhecido como corpo da classe. Os elementos da classe, que estão numerados abaixo, fazem parte do corpo.
  - Construtor – é usado para criar novos objetos.
  - Campos – são variáveis, declaradas dentro da classe (conhecidas como variáveis-membro). Os valores, que estão nos campos, refletem o estado específico do objeto dado, mas apesar disso existem outros tipos de campos, chamados estáticos, que são compartilhados entre todos os objetos (e.g., id aluno).
  - Propriedades – esta é a maneira de descrever as características de uma determinada classe (e.g., get, set).
  - Métodos –blocos de código de programação. Realizam ações particulares e através deles os objetos atingem o seu comportamento. Os métodos executam a lógica de programação implementada (algoritmos) e o tratamento dos dados.

Objetos

- Para poder usar uma determinada classe, primeiro precisamos criar um objeto dela.
- Isso é feito pela palavra reservada **new** em combinação com alguns dos construtores da classe. Isso criará um objeto de uma determinada classe (tipo).
- Se quisermos manipular o objeto recém-criado, teremos que atribuí-lo a uma variável do seu tipo de classe. Ao fazê-lo, nesta variável manteremos a ligação (referência) ao objeto.
- Usando a variável e a notação “ponto”, podemos chamar os métodos e as propriedades do objeto, além de obter acesso aos campos (variáveis-membro).

Objetos - exemplo

```
static void Main()
{
   string firstDogName = null;
   Console.Write("Enter first dog name: ");
   firstDogName = Console.ReadLine();

   // Using a constructor to create a dog with specified name
   Dog firstDog = new Dog(firstDogName);

   // Using a constructor to create a dog wit a default name
   Dog secondDog = new Dog();

   Console.Write("Enter second dog name: ");
   string secondDogName = Console.ReadLine();

   // Using property to set the name of the dog
   secondDog.Name = secondDogName;

   // Creating a dog with a default name
   Dog thirdDog = new Dog();

   Dog[] dogs = new Dog[] { firstDog, secondDog, thirdDog };

   foreach (Dog dog in dogs)
   {
      dog.Bark();
   }
}
```

Modificadores e níveis de acesso (visibilidade)

Em C# existem quatro modificadores de acesso:

- Public
- Private
- Protected
- Internal.

Os modificadores de acesso podem ser usados **apenas** afrente dos seguintes elementos da classe: declaração de classe, campos, propriedades e métodos.

Modificadores e níveis de acesso (visibilidade)

Em C# existem quatro modificadores de acesso:

- Public - este elemento pode ser acedido em todas as classes
- Private - define o nível mais restrito de visibilidade da classe e elementos. O modificador private é usado para indicar que o elemento, para o qual é emitido, não pode ser acedido a partir de outra classe (exceto a classe, na qual está definido)
- Protected - acessível dentro da sua classe e por instâncias de classe derivadas.
- Internal - usado para limitar o acesso aos elementos da classe apenas para ficheiros do mesmo projeto no Visual Studio

Os modificadores de acesso podem ser usados **apenas** afrente dos seguintes elementos da classe: declaração de classe, campos, propriedades e métodos.

Mais sobre as classes

Para a criação dos nomes das classes existem os seguintes padrões comuns:

- Os nomes das classes começam com letra maiúscula e as restantes letras são minúsculas.
- Se o nome da classe for composto por várias palavras, cada palavra começa com letra maiúscula, sem separador a ser usado (convenção PascalCase).
- Para o nome das classes, geralmente são usados substantivos.
- Recomenda-se que o nome da classe esteja em inglês.

A palavra reservada ”this"

- É usada para referenciar o objeto atual (i.e., **instância atual da classe)**
- A palavra reservada pode ser considerada como endereço (referência) com o qual acedemos os elementos (campos, métodos, construtor) da própria classe:

```
public class Employee
{
    private string alias;
    private string name;

    public Employee(string name, string alias)
    {
        // Use this to qualify the members of the class
        // instead of the constructor parameters.
        this.name = name;
        this.alias = alias;
    }
}
```

A palavra reservada ”this"

- É usada para referenciar o objeto atual (i.e., **instância atual da classe)**
- A palavra reservada pode ser considerada como endereço (referência) com o qual acedemos os elementos (campos, métodos, construtor) da própria classe:

```
public class Employee
{
    private string alias;
    private string name;

    public Employee(string name, string alias)
    {
        // Use this to qualify the members of the class
        // instead of the constructor parameters.
        this.name = name;
        this.alias = alias;
    }
}
```

Inner Classes (Nested Classes)

Uma classe interna (aninhada) é chamada de classe que é declarada dentro do corpo de outra classe. A classe que inclui a classe interna é chamada de classe externa.

Os principais motivos para declarar uma classe em outra são:

- Organizar melhor o código ao trabalhar com objetos do mundo real, entre os quais têm uma relação especial e um não pode existir sem o outro.
- Para ocultar uma classe noutra classe, para que a classe interna não possa ser usada fora da classe que a envolveu.

Em geral, as classes internas raramente são usadas, pois complicam a estrutura do código e aumentam os níveis aninhados.

Inner Classes (Nested Classes)

```
public class OuterClass
{
   private string name;

   private OuterClass(string name)
   {
      this.name = name;
   }

   private class NestedClass
   {
      private string name;
      private OuterClass parent;

      public NestedClass(OuterClass parent, string name)
      {
         this.parent = parent;
         this.name = name;
      }

      public void PrintNames()
      {
         Console.WriteLine("Nested name: " + this.name);
         Console.WriteLine("Outer name: " + this.parent.name);
      }
   }

   static void Main()
   {
      OuterClass outerClass = new OuterClass("outer");
      NestedClass nestedClass = new
         OuterClass.NestedClass(outerClass, "nested");
      nestedClass.PrintNames();
   }
}
```

TPC

Implementar um programa que simule o canil municipal.

- Limite de animais a acolher (considerar apenas cães e gatos)
- Permitir registar um animal acolhido
- Permitir registar a adoção de um animal

Princípios Programação Orientada a Objetos

Princípios fundamentais da POO:

- Encapsulamento
- Herança
- Abstração
- Polimorfismo

Princípios Programação Orientada a Objetos

Herança

- Permite que uma classe "herde" (comportamento ou características) de outra classe mais geral.
- Por exemplo, um leão pertence à família dos gatos (Felidae). Todos os gatos têm quatro patas, são predadores e caçam suas presas.
- Esta funcionalidade pode ser codificada uma vez na classe Felidae e todos os predadores podem reutilizá-la – Tiger, Puma, Bobcat, etc.

Princípios Programação Orientada a Objetos

Herança

```
public class Felidae
{
  private bool male;

  // This constructor calls another constructor
  public Felidae() : this(true)
  {}

  // This is the constructor that is inherited
  public Felidae(bool male)
  {
    this.male = male;
  }

  public bool Male
  {
    get { return male; }
    set { this.male = value; }
  }
}
```

```
public class Lion : Felidae
{
  private int weight;


  public Lion(bool male, int weight)
  {
    this.weight = weight;
  }

  public int Weight
  {
    get { return weight; }
    set { this.weight = value; }
  }
}
```

Princípios Programação Orientada a Objetos

Abstração

- Abstração significa trabalhar com algo sem saber como funciona internamente.
- Omite todos os detalhes de um determinado objeto que não nos dizem respeito e usa apenas os detalhes que são relevantes para o problema que estamos resolvendo.
  - Por exemplo, nas configurações de hardware, existe uma abstração chamada "dispositivo de armazenamento de dados" que pode ser um disco rígido, pendrive ou drive de CD-ROM. Cada um deles funciona de maneira diferente internamente, mas, do ponto de vista do sistema operacional, é usado da mesma maneira – armazena ficheiros e pastas.

Princípios Programação Orientada a Objetos

Abstração

- A abstração permite definir uma interface para os nossos sistemas, ou seja, definir todas as tarefas que o programa é capaz de executar e os seus dados de entrada e saída.
- Assim, podemos fazer alguns programas pequenos, cada um lidando com uma tarefa específica. Isto permite ter uma maior flexibilidade na integração desses pequenos programas e muito mais oportunidades de reutilização de código. Os pequenos subprogramas são chamados de componentes.
- Esta abordagem para escrever programas é amplamente adotada, pois permite reutilizar não apenas objetos, mas também subprogramas inteiros.

Princípios Programação Orientada a Objetos

Abstração

- Classes abstratas ou classes base abstratas não podem ser usadas para instanciar objetos.
- As classes abstratas são muito gerais para criar objetos reais - elas **apenas** especificam o que é comum entre as classes derivadas.
- Uma classe abstrata normalmente contém um ou mais métodos abstratos, que possuem a palavra-chave abstract na sua declaração.
- Uma classe que contém métodos abstratos deve ser declarada como uma classe abstrata mesmo que contenha métodos concretos (não abstratos).
- Métodos abstratos não fornecem implementações.
- Construtores e métodos estáticos não podem ser declarados abstratos.

Princípios Programação Orientada a Objetos

```
public abstract class Pet
  {
     private string myName;
     public Pet(string name)
     {
        myName = name;
     }
     public string getName()
     {
        return myName;
     }
     public abstract string speak();
  }
```

```
public class Cat : Pet
{
  public Cat(string name): base(name)
    {   }


    public override string speak()
   {
    return "meow";
    }
}
```

Princípios Programação Orientada a Objetos

Abstração

- Podemos usar classes abstratas para declarar variáveis que podem conter referências a objetos de qualquer classe concreta derivada dessas classes abstratas.
- Podemos usar essas variáveis para manipular polimorficamente objetos da classe derivada e invocar métodos estáticos declarados nessas classes abstratas.

Princípios Programação Orientada a Objetos

Interfaces

- As interfaces definem as maneiras pelas quais as pessoas e os sistemas podem interagir uns com os outros.
- Uma interface descreve um conjunto de métodos que podem ser chamados no objeto – para instruí-lo, por exemplo, a realizar alguma tarefa ou retornar alguma informação.
- Uma declaração de interface começa com a palavra-chave interface. Todos os membros da interface são declarados implicitamente como públicos e abstratos. Uma interface normalmente especifica o comportamento que uma classe implementará.
- Uma interface pode estender uma ou mais outras interfaces para criar uma interface mais elaborada que outras classes podem implementar.

Princípios Programação Orientada a Objetos

Interfaces

- Quando temos várias classes que precisam de implementar o mesmo método, é aconselhável usar interfaces.
- Interfaces são como classes (propriedades, métodos e eventos), mas não contêm código. A codificação será feita na classe que implementa a interface.

Princípios Programação Orientada a Objetos

Interfaces

- Uma interface só pode declarar métodos e constantes.
- Uma interface é normalmente usada quando classes não relacionadas precisam de partilhar métodos comuns para que possam ser processadas polimorficamente
- Podemos criar uma interface que descreva a funcionalidade desejada e, em seguida, implementar essa interface em qualquer classe que exija essa funcionalidade.
- Uma interface geralmente é usada no lugar de uma classe abstrata quando não há implementação padrão a ser herdada - ou seja, nenhum campo e nenhuma implementação de método padrão.

Princípios Programação Orientada a Objetos

Interfaces

```
public interface IMoveable
{
void move ();

void reverse ();
}
```

```
public class Player: IMoveable
{
. . .
public void move ()
{
posX += 1;
posY += 1;
}
public void reverse ()
{
posX -= 1;
posY -= 1;
}
```

Princípios Programação Orientada a Objetos

Encapsulamento

- É chamado de "ocultação de informações". Um objeto deve fornecer apenas as informações essenciais para a sua manipulação, sem os detalhes internos.
  - Uma Secretária que usa um computador só conhece o ecrã, teclado e rato. Oresto está escondido internamente na “caixa”.
- O programador deve decidir o que deve ser escondido e o que não. Quando programamos, devemos definir como privado todos os métodos ou campos que outras classes não podem aceder.

Princípios Programação Orientada a Objetos

Polimorfismo

- O polimorfismo permite tratar objetos de uma classe derivada como objetos da sua classe base. Por exemplo, grandes felinos (classe base) capturam suas presas (um método) de maneiras diferentes.
- O polimorfismo permite tratar um gato de um tamanho aleatório como um grande felino e mandá-lo ”caçar", independentemente do seu tamanho exato.
- O polimorfismo pode ter uma forte semelhança com a abstração, mas está principalmente relacionado à substituição de métodos em classes derivadas, a fim de alterar o comportamento original herdado da classe base.

Princípios Programação Orientada a Objetos

Polimorfismo

- Métodos Virtuais
  - Um método, que pode ser substituído numa classe derivada, é chamado de método virtual. Os métodos em .NET por padrão não são virtuais. Se quisermos tornar um método virtual, adicionamos a palavra-chave virtual. A classe derivada pode declarar e definir um método com a mesma assinatura.
  - Métodos virtuais são usados quando esperamos que classes derivadas alterem ou complementem algumas das funcionalidades herdadas.
  - Os métodos virtuais são importantes para a substituição de métodos, que está na base do polimorfismo.


qualificar
TAL
A <PROGRAMAÇÃ