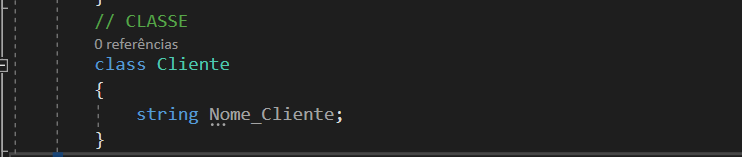
**C# Orientado a objetos**

**Programação Orientada a Objetos**

Programação Orientada a Objetos (também conhecida pela sua sigla POO) é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de 'objetos'.

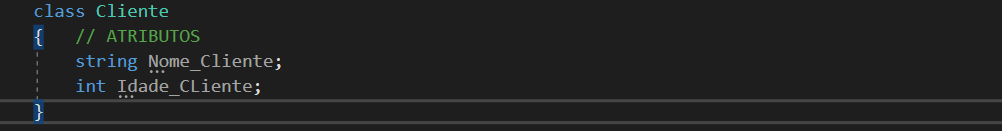
**Classe**

É uma descrição que abstrai um conjunto de objetos com características similares. Mais formalmente, é um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por objetos.



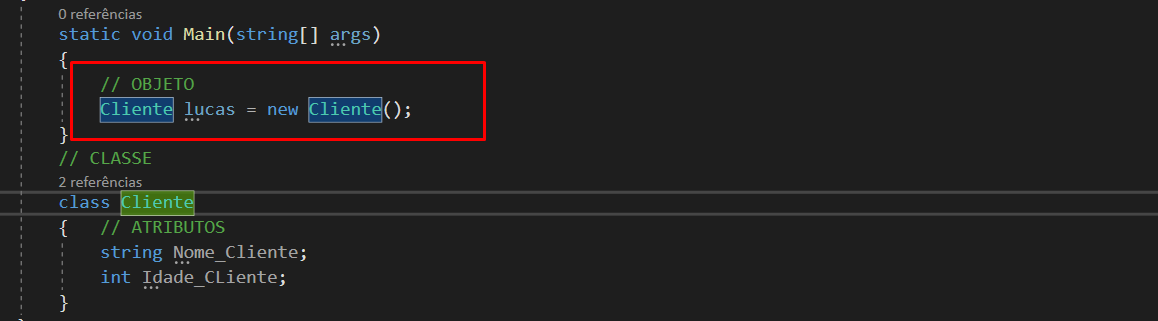
**Atributo**

Os Atributos em Programação Orientada a Objetos são os elementos que definem a estrutura de uma classe. Os atributos também são conhecidos como variáveis de classe, e podem ser divididos em dois tipos básicos: atributos de instância e de classe. Os valores dos atributos de instância determinam o estado de cada objeto.



**Objeto**

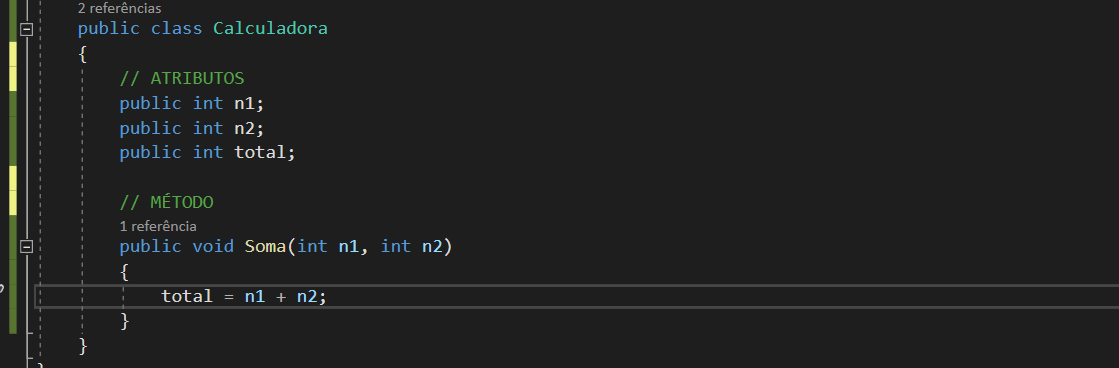
Ele é composto por propriedades, métodos e eventos da classe. Um objeto sempre será do tipo de referência, ou seja, ele só armazena o endereço dos dados.



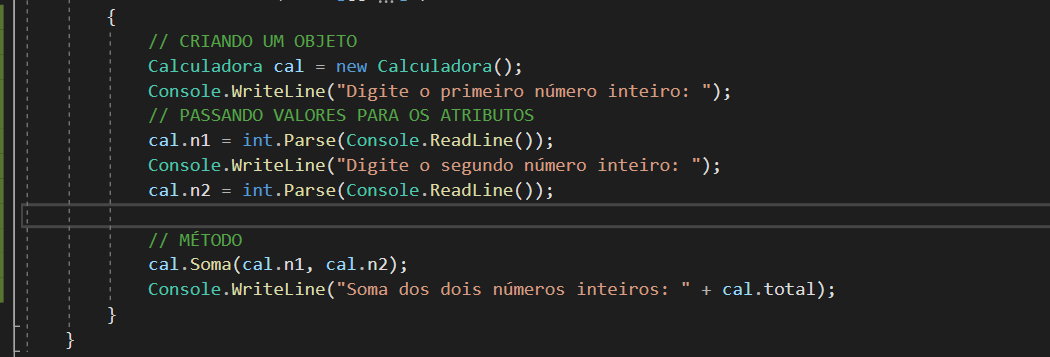
**Métodos**

É uma função é uma sub-rotina (ou procedimento ou função) associada a um objeto, e que possui acesso aos seus dados, as variáveis membro. Ele é executado por um objeto ao receber uma mensagem, como (cadastrar, excluir, alterar).

Primeiro foi criado uma classe com o nome de calculadora, com três atributos e um método do tipo void, ou seja, que não tem retorno ele apenas realiza a adição dos números e atribui ao atributo total.

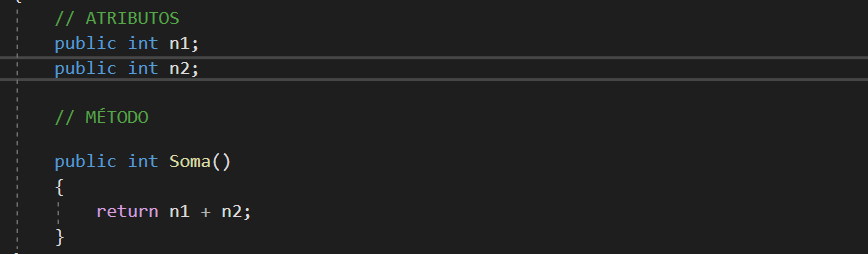


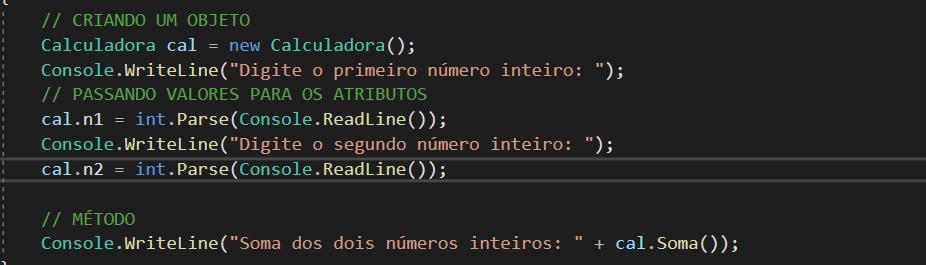
Em seguida foi criado um objeto, que é composto pelos atributos e métodos da classe calculadora



**Utilizando o mesmo método com return**

Diferente do método do tipo void, ele não recebe parâmetros de entrada, apenas retorna a soma dos dois números, sem a necessidade de criar outra variável para receber os valores n1 e n2.





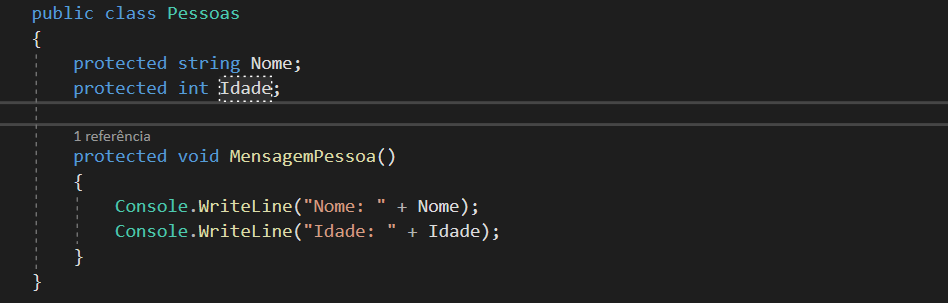
**Quando utilizar cada um?**

O método do tipo void é usado para quando queremos salvar algum valor em uma variável. Já o método do tipo return, como o próprio nome já diz, ele é utilizando quando já queremos retornar algo sem a necessidade de salvar em alguma variável.

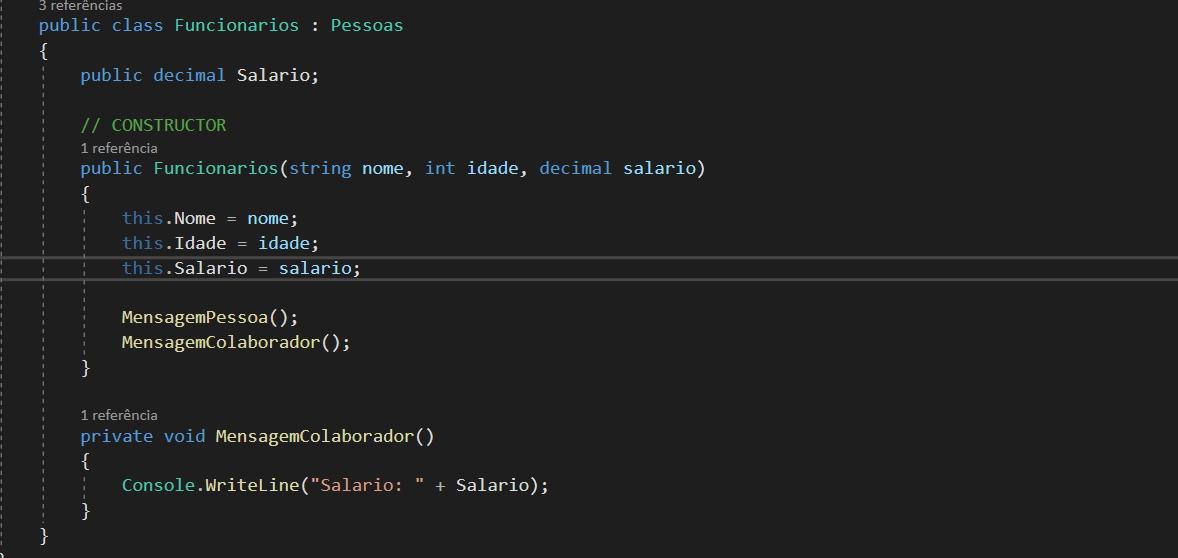
**Herança**

A herança é um princípio próprio à programação orientada a objetos (POO) que permite criar uma nova classe a partir de uma já existente. A principal vantagem da herança é a capacidade para definir novos atributos e métodos para a subclasse, que se somam aos atributos e métodos herdados.

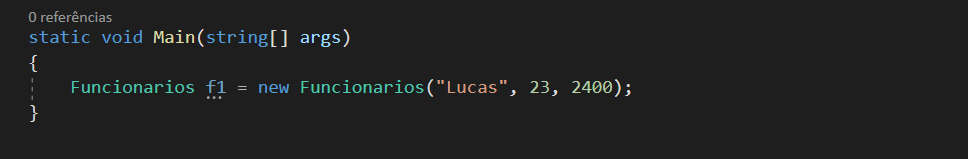
Exemplo: foi criado uma classe pessoa como modelo para as outras classes, com dois atributos e um método que imprime seu nome e idade.



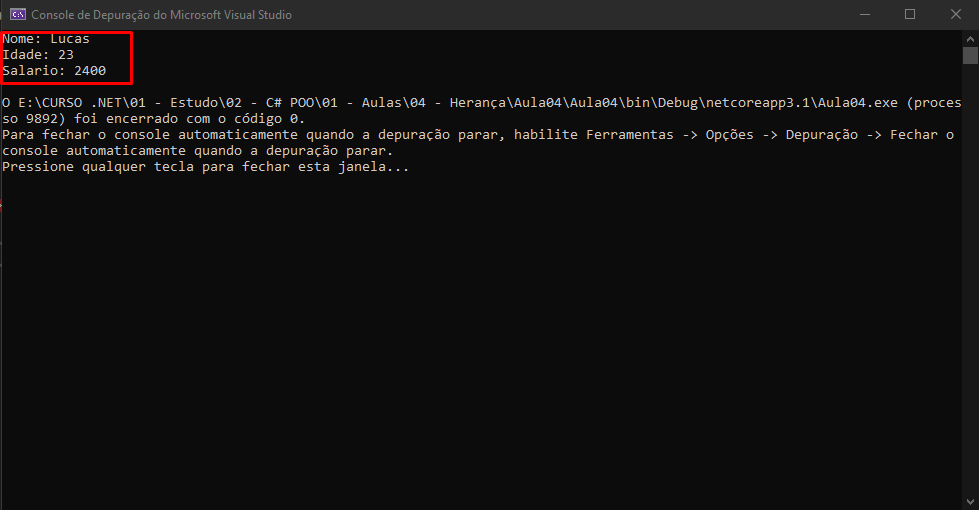
Em seguida foi criado uma segunda classe chamada Funcionários que herda da classe pessoas, ou seja, ela também tem os atributos e métodos criados na classe pessoas.



Na imagem acima foi utilizado um método do tipo construtor para iniciar os atributos com valores, que serão passados quando o objeto é instanciado.



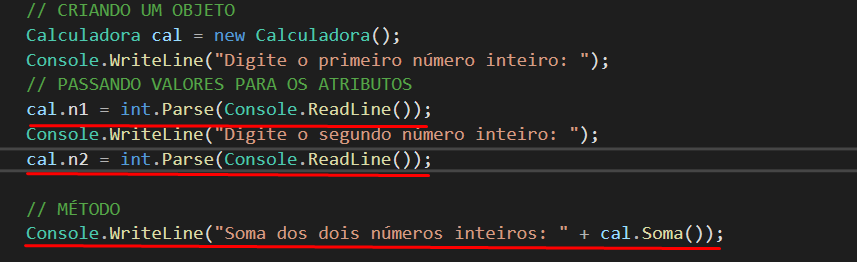
Instanciando um objeto já com valores nos seus atributos, no resultado podemos ver que é impresso o nome e idade, sendo da classe pessoas mesmo que criamos um objeto da classe funcionários, pois a classe funcionários consequentemente herda todos os atributos e métodos da classe pessoas.



**Encapsulamento**

Resumidamente o encapsulamento é utilizar uma classe dentro de outra classe e poder utilizar seus atributos, métodos, eventos, esse é o conceito de encapsulamento.

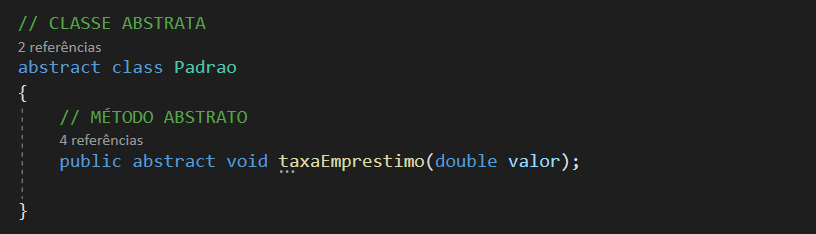
Exemplo: na classe principal está sendo utilizado os atributos e métodos de outra classe do programa, resumidamente colocando uma dentro da outra igual um remédio dentro de uma capsula.



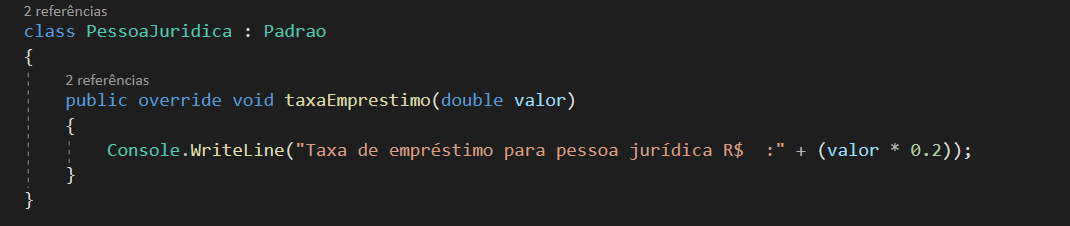
**Abstração**

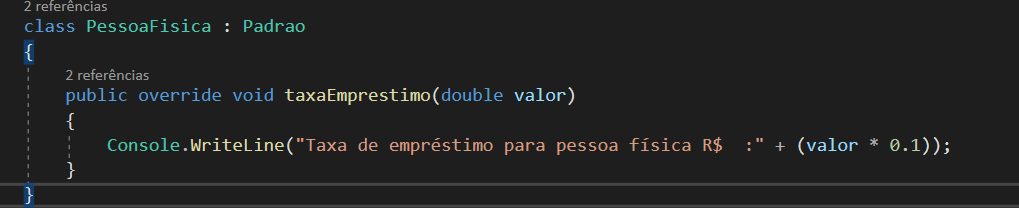
A classe abstrata não possui implementação, ela é utilizada como modelo para as subclasses, assim como os métodos abstratos o corpo dele não tem implementação, ele é um método vazio sendo implementado nas subclasses.

Exemplo: foi criado uma classe e um método abstrato para calcular a taxa de empréstimos para pessoas físicas e jurídicas.



Observe que nas duas imagens abaixo ambas herdam da classe Padrão, consequentemente tendo os métodos da classe herdada, porém o método herdado é do tipo abstrato, por padrão ele é criado sem implementação para ser implementados nas subclasses.



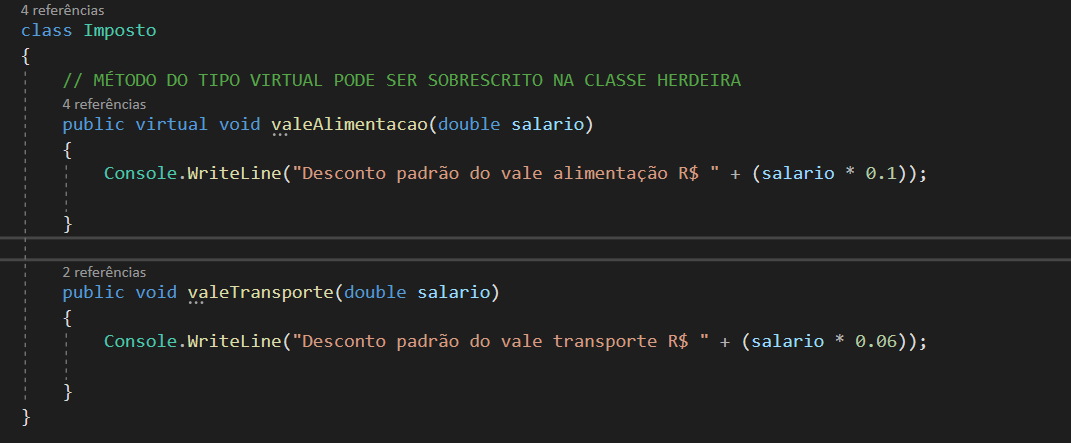


Ambas as subclasses tem métodos para calcular a taxa de empréstimos desta forma está sendo reescrito o método abstrato da classe abstrata.

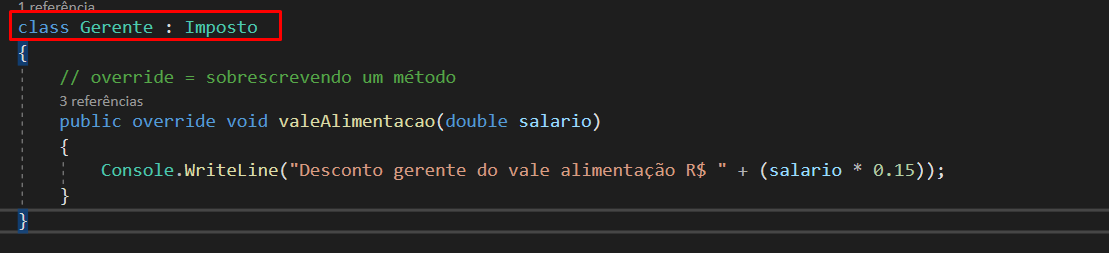
**Polimorfismo**

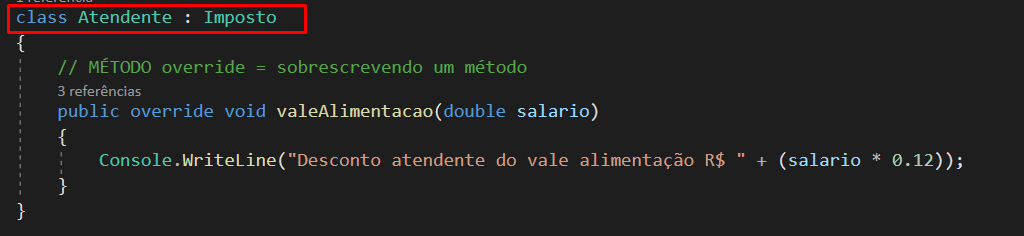
O Polimorfismo é um mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução. Com o Polimorfismo, os mesmos atributos e objetos podem ser utilizados em objetos distintos, porém, com implementações lógicas diferentes.

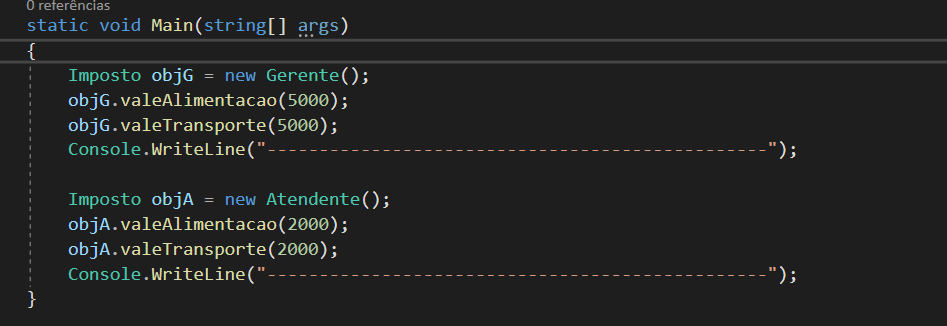
Exemplo: foi criado uma classe imposto para mostrar o desconto do vale alimentação e do vale transporte.



As duas imagens abaixo ambas herdam da classe imposto, contentando os seus métodos da classe herdada, diferente dos métodos abstratos mostrados anteriormente. Os métodos da classe Imposto já são implementados no momento da criação da classe, porém podem ser alterados nas subclasses.







**Modificadores de acesso**

São as palavras-chaves usadas para especificar a acessibilidade de uma classe, método e atributos.

public = Atributos e métodos visíveis para todas as classes.

private = Atributos e métodos visíveis apenas onde foi criado.

protected = Atributos e métodos visíveis em classes onde são criados ou herdados.

Internal = Atributos e métodos visíveis apenas para o mesmo namespace onde foi criado.

Exemplo:

public string nome; -> fica visível quando o objeto é instanciado.

private double valor; -> não fica visível quando o objeto é instanciado.

**Membro estático**

Membros estáticos são membros de dados (variáveis) ou métodos que pertencem a uma classe estática ou não estática, em vez de objetos da classe. Os membros estáticos sempre permanecem os mesmos, independentemente de onde e como são usados. Como os membros estáticos estão associados à classe, não é necessário criar uma instância dessa classe para invocá-los.

**Construtor**

É um método com o mesmo nome da classe, onde ele é executado assim que instancia um objeto.

**Sobrecarga**

Sobrecarregar métodos significa ter vários métodos com nomes iguais, mas assinaturas diferentes. A assinatura de um método no C# é dada pela junção do seu nome e seus parâmetros de entrada.

**This**

É usado para referenciar um atributo da classe.

**GET e SET**

Os métodos GET e SET são técnicas padronizadas para gerenciamento sobre o acesso dos atributos. Nesses métodos determinamos quando será modificados os atributos e o acesso ao mesmo, tornando o controle e modificações mais práticas e limpas, sem, contudo, precisar alterar assinatura do método usado para acesso ao atributo.

* Set = envia uma informação para um atributo
* Get = obtém a informação enviada