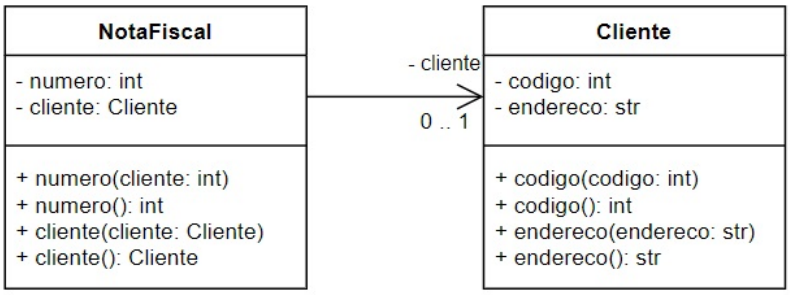
Associação

- Relação permanente entre dois objetos
- Definida pela existência de um atributo do tipo da classe associada
- Atributo guarda uma referência para o objeto associado

Vamos ver um exemplo prático:



> Nesse exemplo, a classe **NotaFiscal** está associada à classe **Cliente** por meio do atributo *- cliente*. Em outras palavras, objetos da classe **NotaFiscal** podem possuir uma referência para um objeto da classe **Cliente**. É interessante notar que no atributo *- cliente* da classe **NotaFiscal** não fica guardado o código do cliente, mas sim uma referência para uma instância da classe cliente.

Vamos implementar a classe Cliente

De acordo com o diagrama de classes acima, a classe cliente possui dois atributos: codigo e endereco. Ambos são privados, pois têm o símbolo "**-**" na frente do nome do atributo. Note que os parâmetros dos métodos têm a indicação do tipo do dado. A indicação desse tipo não torna o tipo obrigatório, mas instrui o programador que for utilizar o método do que passar como parâmetro.

Construtor e atributos

O construtor da classe é o método `def init` que recebe como parâmetros `self, codigo: int, endereco: str` :

- `self` representa o próprio objeto que está sendo instanciado. É nesse objeto que são criados os atributos. Todos os outros métodos da classe também recebem esse parâmetro, para poder manipular os valores dos atributos.
- `codigo: int, endereco: str` são atributos simples da própria classe **Cliente**, por isso são criados dois atributos no *self*, que representa o próprio objeto: `self.codigo = codigo` e `self.endereco = endereco` .

São criados dois métodos especiais para cada atributo:

- *@property* que permite recuperar o dado do atributo
- *@NOME_DO_ATRIBUTO.setter* que permite alterar o dado do atributo

Checagem de tipos:

Como exemplo, incluímos um teste para verificar o tipo de dado que está sendo enviado utilizando `isinstance`. Da mesma forma, incluímos `-> int` e `-> str` como informação do tipo de retorno nos métodos que retornam esses tipos. Essa informação ajuda o programador que irá utilizar esses métodos no futuro.

In [41]:

```
class Cliente:

  def __init__(self, codigo: int, endereco: str):
    if isinstance(codigo, int):
      self.__codigo = codigo
    if isinstance(endereco, str):
      self.__endereco = endereco

  @property
  def codigo(self) -> int:
    return self.__codigo

  @codigo.setter
  def codigo(self, codigo: int):
    if isinstance(codigo, int):
      self.__codigo = codigo

  @property
  def endereco(self) -> str:
    return self.__endereco

  @endereco.setter
  def endereco(self, endereco: str):
    if isinstance(endereco, str):
      self.__endereco = endereco
```

Agora a classe NotaFiscal

A classe **NotaFiscal** também possui dois atributos: numero e cliente. O atributo - *numero* é simples do tipo *int* e o atributo - *cliente* é do tipo **Cliente**.

Construtor e atributos

O construtor da classe é o método `def __init__` que recebe como parâmetros `self, numero: int, cliente: Ciente` :

- `self` representa o próprio objeto que está sendo instanciado, da mesma forma que na classe anterior. É nesse objeto que são criados os atributos. Aqui também todos os outros métodos da classe também recebem esse parâmetro, para poder manipular os valores dos atributos.
- `numero: int` é um atributo simples da própria classe **NotaFiscal**, por isso é criado um atributo no *self* que representa o próprio objeto: `self.__numero = numero` .
- `cliente: Cliente` é um atributo do tipo da classe **Cliente**. Dentro deste atributo será armazenado uma referênci apara um cliente que já vem instanciado como parâmetro: `self.__clinte = cliente` .

Também são criados dois métodos especiais para cada atributo:

- *@property* que permite recuperar o dado do atributo
- *@NOME_DO_ATRIBUTO.setter* que permite alterar o dado do atributo

Checagem de tipos:

E também incluímos um teste para verificar o tipo de dado que está sendo enviado utilizando `isinstance` para garantir, por exemplo, que no cliente venha realmente um objeto da classe **Cliente** como parâmetro, evitando assim erros posteriores.

In [42]:

```
class NotaFiscal:

  def __init__(self, numero: int, cliente: Cliente):
    if isinstance(numero, int):
      self.__numero = numero
    if isinstance(cliente, Cliente):
      self.__cliente = cliente

  @property
  def numero(self) -> int:
    return self.__numero

  @numero.setter
  def numero(self, numero: int):
    if isinstance(numero, int):
      self.__numero = numero

  @property
  def cliente(self) -> Cliente:
    return self.__cliente

  @cliente.setter
  def cliente(self, cliente: Cliente):
    if isinstance(cliente, Cliente):
      self.__cliente = cliente
```

Agora vamos brincar um pouco com os conceitos

Primeiro vamos instanciar objetos das duas classes:

In [60]:

```
# Instanciando:

um_cliente = Cliente(123, "Rua Geral, sem nº")
segundo_cliente = Cliente(456, "Avenida Importante, nº 456")
terceiro_cliente = Cliente(789, "Rua Mais Importante, nº 789")

# colocando os clientes em uma lista de clientes:
clientes = [um_cliente, segundo_cliente, terceiro_cliente]

# O cliente 123 fez uma compra
uma_nota_fiscal = NotaFiscal(11, um_cliente)

# O cliente 456 fez duas compras
outra_nota_fiscal = NotaFiscal(22, segundo_cliente)
mais_outra_nota_fiscal = NotaFiscal(33, segundo_cliente)

# colocando as notas fiscais em uma lita de notas:
notas = [uma_nota_fiscal, outra_nota_fiscal, mais_outra_nota_fiscal]


# Imprimindo os valores dos atributos dos clientes:
print("Cliente:", um_cliente.codigo, um_cliente.endereco)
print("Cliente:", segundo_cliente.codigo, segundo_cliente.endereco)
print("Cliente:", terceiro_cliente.codigo, terceiro_cliente.endereco)

# Imprimindo os valores dos atributos das notas
print("Nota Fiscal:", uma_nota_fiscal.numero)
print("Nota Fiscal:", outra_nota_fiscal.numero)
print("Nota Fiscal:", mais_outra_nota_fiscal.numero)
```

```
Cliente: 123 Rua Geral, sem nº
Cliente: 456 Avenida Importante, nº 456
Cliente: 789 Rua Mais Importante, nº 789
Nota Fiscal: 11
Nota Fiscal: 22
Nota Fiscal: 33
```

Agora obtendo os dados dos clientes e notas que estão nas listas:

In [61]:

```
# Clientes

for cliente in clientes:
    print("Cliente:", cliente.codigo, cliente.endereco)

# Notas

for nota in notas:
    print("Nota:", nota.numero)
```

```
Cliente: 123 Rua Geral, sem nº
Cliente: 456 Avenida Importante, nº 456
Cliente: 789 Rua Mais Importante, nº 789
Nota: 11
Nota: 22
Nota: 33
```

Mas, como obter os dados dos clientes de uma determinada Nota?

Deve-se acessar por meio do objeto que está associado no atributo:

In [68]:

```
# Mostrando os dados de um objeto cliente de uma nota:

print("Cliente da Nota:", uma_nota_fiscal.numero)
print("Codigo do Cliente:", uma_nota_fiscal.cliente.codigo)
print("Endereço do Cliente:", uma_nota_fiscal.cliente.endereco)
```

```
Cliente da Nota: 11
Codigo do Cliente: 123
Endereço do Cliente: Rua Geral, sem nº
```

In [57]:

```
# Pegando o código do cliente e convertendo para inteiro:

codigo_cliente = int(input("Informe o código do cliente:"))

# Percorrendo as notas e verificando qual é o cliente

print("Notas do Cliente:", codigo_cliente)
for nota in notas:
    if nota.cliente.codigo == codigo_cliente:
        print("> Nota:", nota.numero)
```

```
Informe o código do cliente: 456

Notas do Cliente: 456
> Nota: 22
> Nota: 33
```

Entendendo as referências para os objetos

O que acontece com as notas se alterarmos o endereço de um dos clientes.

Vamos testar ... quando for requisitado, informe um novo endereço para o cliente:

In [64]:

```
# Pegando novo endereço para o cliente 456 (objeto: segundo_cliente)

novo_endereco = input("Informe o novo endereço do cliente 456:")
segundo_cliente.endereco = novo_endereco
```

```
Informe o novo endereço do cliente 456: NOVO ENDERECO
```

O que será que acontece com o endereço dos clientes que estão na nota?

In [65]:

```
# Mostrando os endereços dos clientes das notas

for nota in notas:
    print("Nota número:", nota.numero)
    print("> Cliente da Nota:", nota.cliente.codigo)
    print("> Endereço do Cliente da Nota:", nota.cliente.endereco)
```

```
Nota número: 11
> Cliente da Nota: 123
> Endereço do Cliente da Nota: Rua Geral, sem nº
Nota número: 22
> Cliente da Nota: 456
> Endereço do Cliente da Nota: NOVO ENDERECO
Nota número: 33
> Cliente da Nota: 456
> Endereço do Cliente da Nota: NOVO ENDERECO
```

Olha que interessante!!! Os endereços dos "clientes" 456 que estão nas notas também mudaram!

Isso ocorre porque, na verdade, os clientes não estão nas notas, mas só uma referência para o objeto que está na memória.

Vamos testar isso utilizando o comando `id(objeto)` que mostra um número que equivale ao identificador do objeto na memória. Essa é realmente a IDENTIDADE do objeto.

veja quais são os identificadores do objeto `outro_cliente` e os identificadores dos objetos clientes associados às notas fiscais:

In [67]:

```
# Mostrando o identificados do cliente outro_cliente:

print("Identificador do cliente 456:", id(segundo_cliente))

# Percorrendo as notas e mostrando o identificador dos clientes

print("Notas do Cliente: 456")
for nota in notas:
    if nota.cliente.codigo == 456:
        print("Nota:", nota.numero)
        print("> Identificador do cliente na nota:", id(nota.cliente))
```

```
Identificador do cliente 456: 139742669234976
Notas do Cliente: 456
Nota: 22
> Identificador do cliente na nota: 139742669234976
Nota: 33
> Identificador do cliente na nota: 139742669234976
```

Note que é o mesmo identificador. Ou seja, os atributos as notas 22 e 33 associam com o mesmo objeto que está na memória!