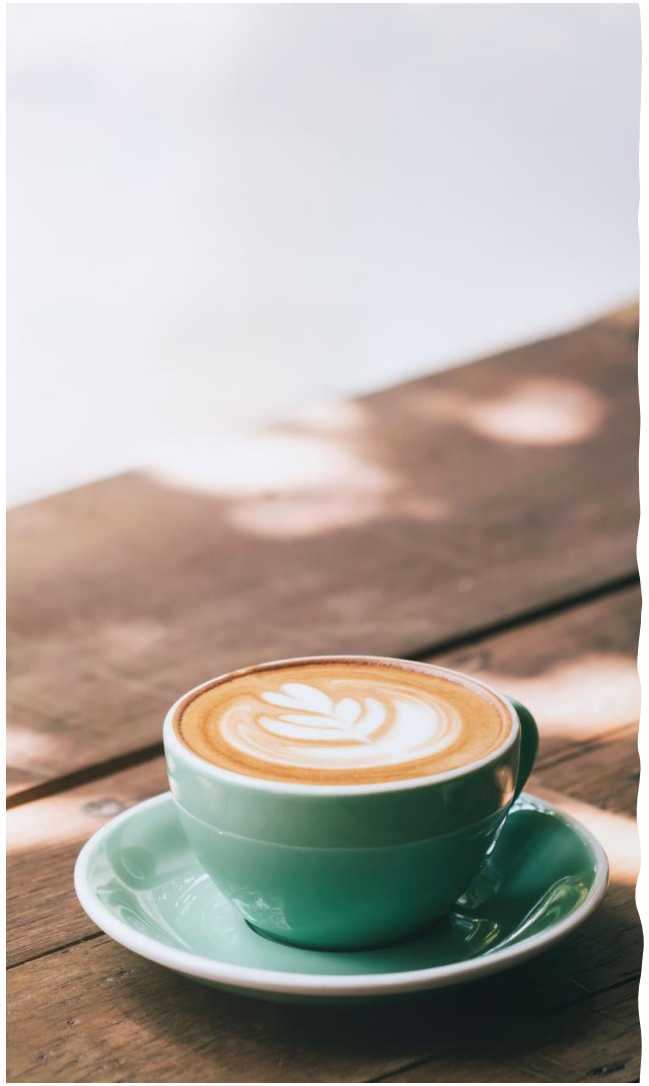


Uma introdução à Orientação por Objetos

Membros Estáticos

Introdução à Generalização

Aula prática anterior: *métodos sobrecarregados*

Métodos sobrecarregados

Métodos com mesmo nome, mas com leve variação em suas assinaturas.

O método a ser executado é identificado por seus parâmetros: tipo e quantidade.

Polimorfismo fraco: método identificável em tempo de compilação.

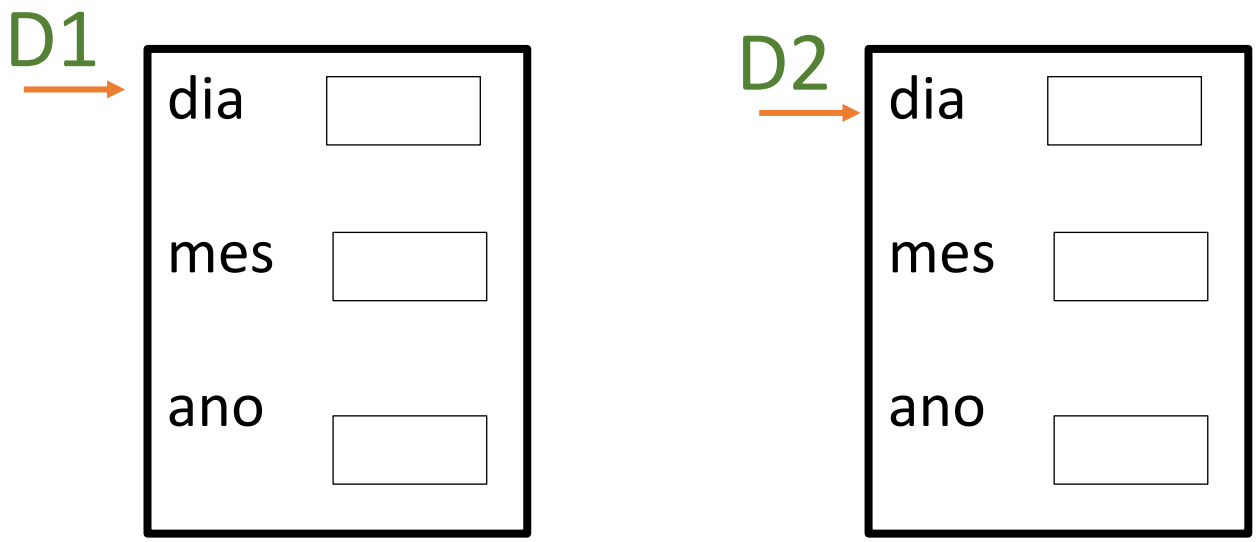
Exemplos:

Salario() : Double
Salario(Integer) : Double
Salario(Double) : Double
Salario(Double, Integer) : Double
Salario(String, Integer) : Double

Aula prática anterior: *construtor* e *destrutor*

Construtores

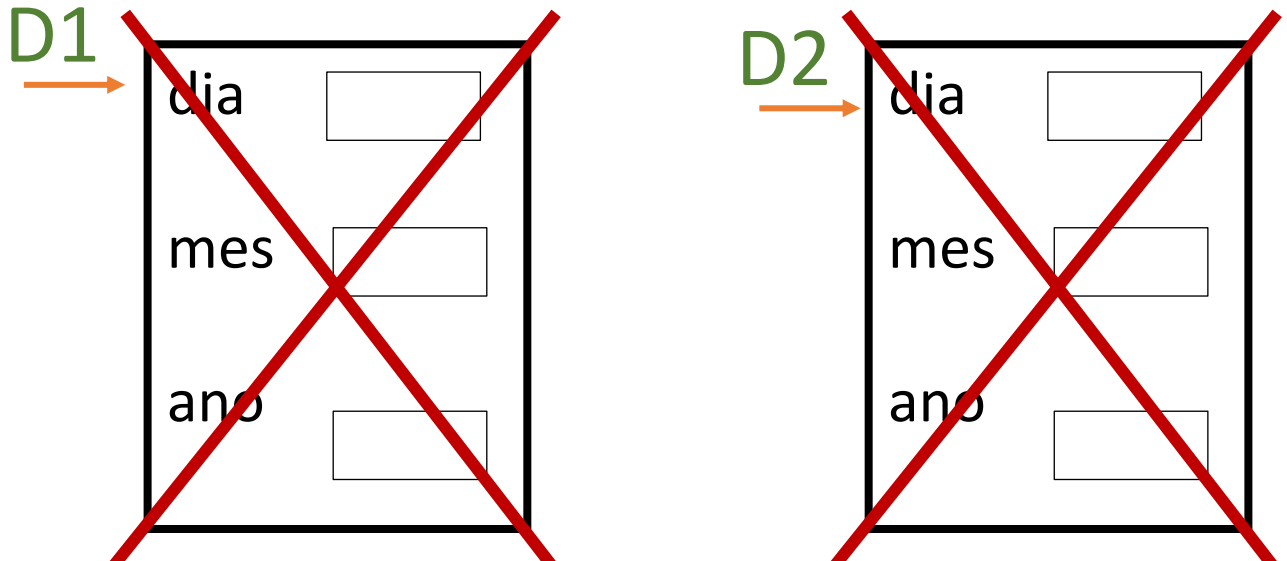
```
Data* D1 = new Data();
Data* D2 = new Data(21,11,2024);
```




```
class Data {
  private :
    int dia;
    int mes;
    int ano;

  public :
    Data() {
      dia=mes=ano= 0;
    }
    Data(int dia, int mes, int ano)  {
      this->setData(dia, mes, ano);
    }
    void setData(int dia, int mes, int ano) {
          setDia(dia);
          setMes(mes);
          setAno(ano);
    }
    ...
};
```

Destrutores

```
void funcao()
{
        Data* D1 = new Data();
        Data* D2 = new Data(21,11,2024);
    ...
    delete D1;
    delete D2;
}
```




```
class Data
{
   private :
      int dia;
      int mes;
      int ano;

   public :
      ~Data()
      {
         printf("\n\aMorri em %p", this);
      }
      Data()
      {
         dia=mes=ano= 0;
      }
      Data(int dia, int mes, int ano)  {
         this->setData(dia, mes, ano);
      }
      ...
};
```

Uma aplicação: *contando objetos*

Contando objetos com Construtor e Destrutor

Observe um potencial serviço que poderia ser implementado:

Uma variável global poderia se encarregar de contar o número de objetos criados no arranjo, incrementando 1 a cada nova instância criada.

Para isto, uma contagem dos objetos pode ser realizada pelos construtores (incrementar) e destrutores (decrementar).

```
int TAM = 0;

class Pessoa{
        private :
                string nome;
                Data   nascimento;
        public :
                Pessoa(){
                        TAM++;
                }
                ~Pessoa(){
                        TAM--;
                }
                ...
};
```

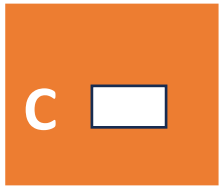
Incorporando propriedades estáticas da classe

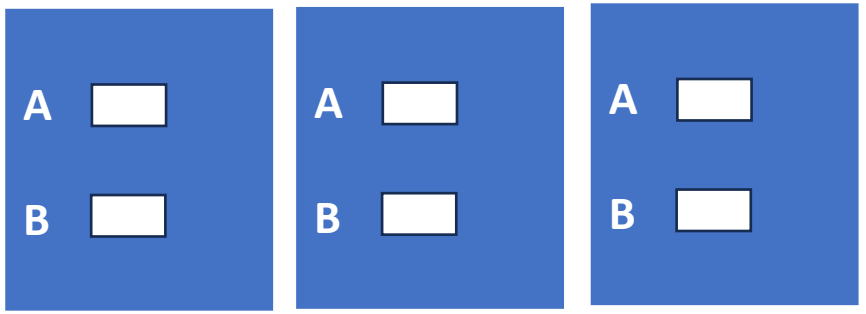
Propriedade estática

Uma propriedade da classe é dita estática se ela pertencer à classe, e não às suas instâncias.

Exemplo: atributo estático

```
class Qualquer {
        private :
                int A;
                int B;
           static int C;
        public :
                ...
};
```



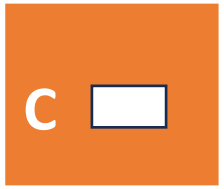


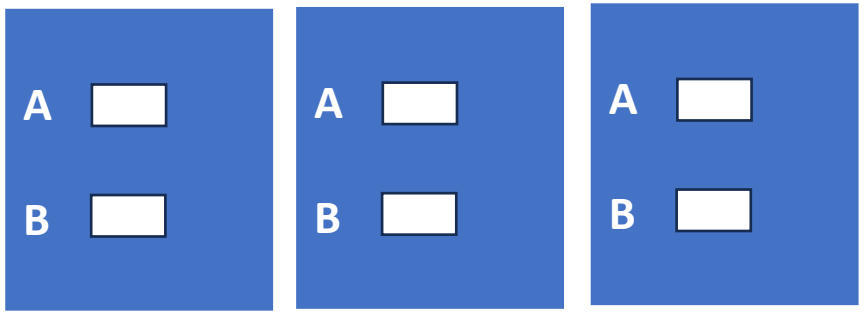

```
Qualquer* Q1, Q2, Q3;
Q1 = new Qualquer;
Q2 = new Qualquer;
Q3 = new Qualquer;
```

Propriedade estática

Exemplo: atributo estático

```
class Qualquer {
        private :
                int A;
                int B;
           static int C;
        public :
                ...
};
int Qualquer::C = 0;
```






```
Qualquer* Q1, Q2, Q3;
Q1 = new Qualquer;
Q2 = new Qualquer;
Q3 = new Qualquer;
```

Exemplo de uso para quantificar o número de instâncias criadas

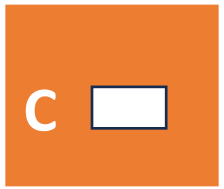
```
int TAM = 0;

class Pessoa{
        private :
                string nome;
                Data   nascimento;
        public :
                Pessoa(){
                        TAM++;
                }
                ~Pessoa(){
                        TAM--;
                }
                ...
};
```

```
int TAM = 0;

class Pessoa{
        private :
                string nome;
                Data   nascimento;
        public :

                Pessoa(){
                        TAM++;
                }

                ~Pessoa(){
                        TAM--;
                }
                ...
};
```

```
class Pessoa{
        private :
                string nome;
                Data   nascimento;
        public :
                static int TAM;
                Pessoa(){
                        TAM++;
                }

                ~Pessoa(){
                        TAM--;
                }
                ...
};
int Pessoa::TAM = 0;
```

Exemplo de atributo estático para contagem

```
class Pessoa{
        private :
                string nome;
                Data   nascimento;
        public :
                static int TAM;
                Pessoa(){
                        TAM++;
                }
                ~Pessoa(){
                        TAM--;
                }
                ...
};

int Pessoa::TAM = 0;
```




```
Pessoa* P1, P2, P3;
P1 = new Pessoa;
P2 = new Pessoa;
P3 = new Pessoa;

cout<< Pessoa::TAM;

delete P2;
cout<< Pessoa::TAM;
```

Exemplo: Contando Cães

Cachorro
- nome : String + quantidade : Integer
+ Cachorro() + Cachorro(String) + fala()

Implemente os métodos de forma que o atributo estático quantidade *represente o número de instâncias da classe* Cachorro

Aplicação do Trabalho Prático Final

Classes Data e Pessoa

Pessoa
- nome : String - nascimento : Data
+ setNome(String) + getNome() : String + setNascimento(Integer, Integer, Integer) + getNascimento() : Data + leiaNome() + escrevaNome() + leiaPessoa() + escrevaPessoa()

Inserir construtores sobrecarregados e destrutor. Contar Pessoas

Data
- dia : Integer - mes : Integer - ano : Integer
+ setDia(Integer) : Boolean + setMes(Integer) : Boolean + setAno(Integer) : Boolean + setData(Integer, Integer, Integer) : Boolean + getDia() : Integer + getMes() : Integer + getAno() : Integer + dataValida() : Boolean + mesExtenso() : String + diasMes() : Integer + escrevaData() + leiaData()

Garanta a corretude das funcionalidades com OO

0 – Sair do programa

1 - Cadastrar uma pessoa

2 - Listar todas as pessoas

3 – Pesquisar por nome

4 – Pesquisar por CPF

5 – Excluir pessoa

6 - Apagar todas as pessoas cadastradas

Generalização (herança)

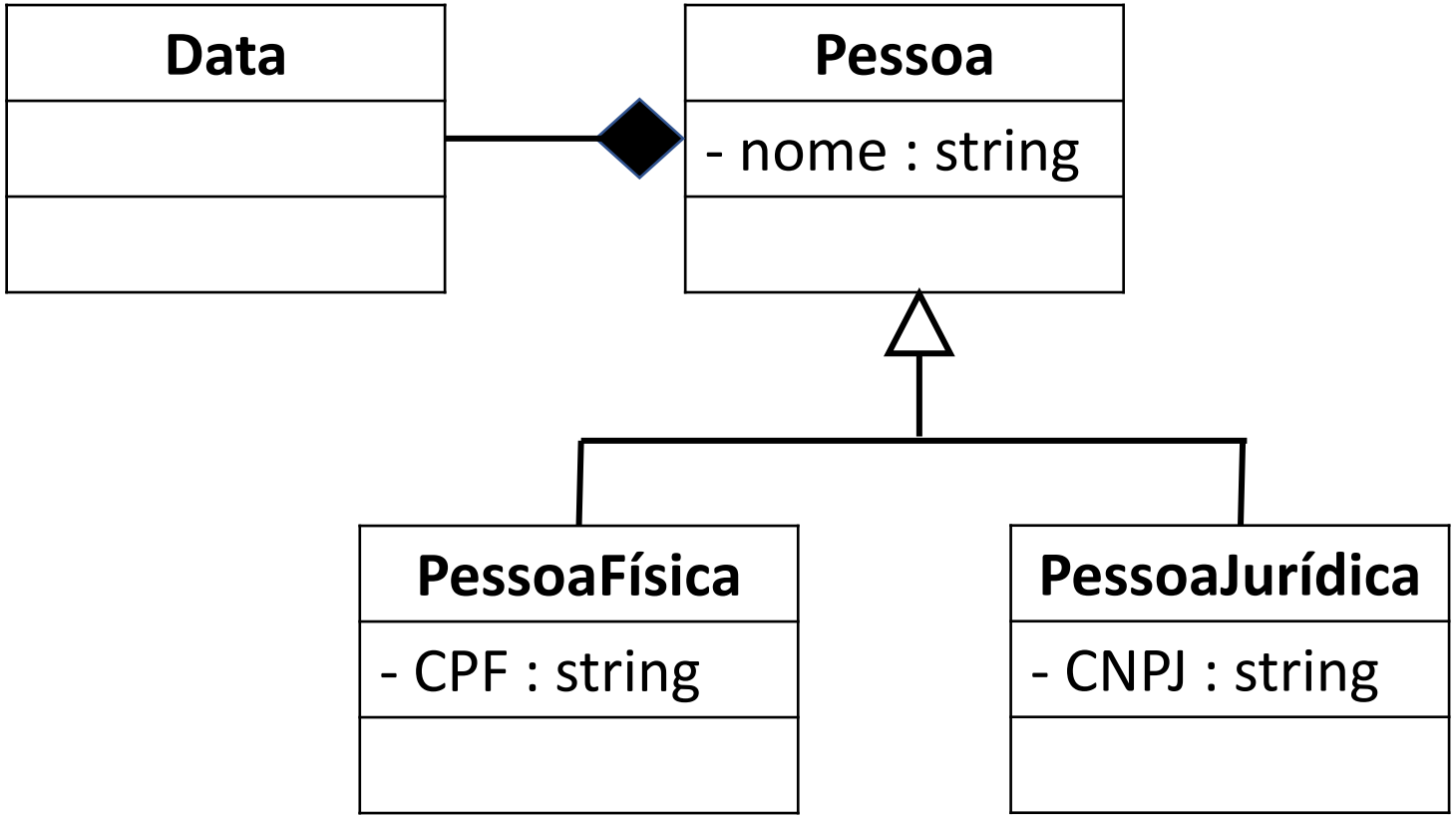
Generalização (herança)

Uma classe (base) pode generalizar as propriedades comuns de outras (derivadas)

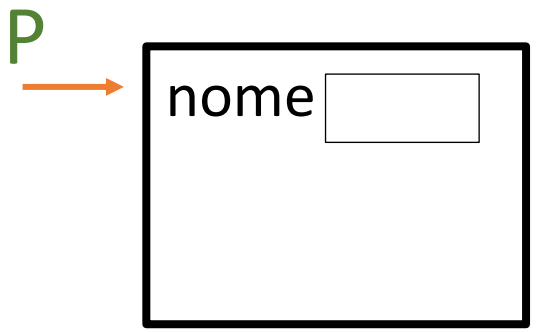
Uma classe (base) pode se especializar em outras (derivadas)

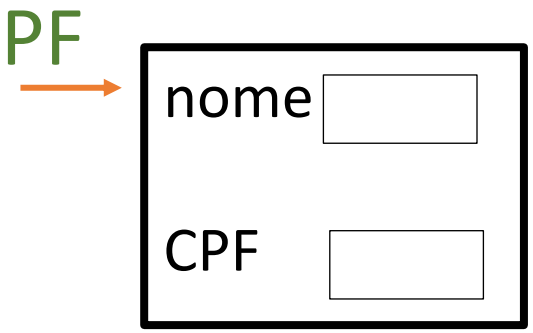
Uma classe (derivada) pode herdar as propriedades descritas em outra (base)

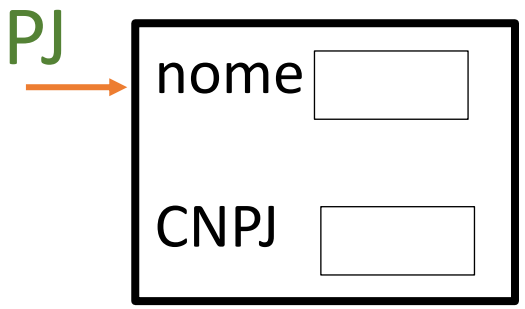
Relacionamento do tipo: ***is a***



```
Pessoa*          P  = new Pessoa();
PessoaFisica*    PF = new PessoaFisica();
PessoaJuridica*  PJ = new PessoaJuridica();
```








```
class Pessoa
{
   private : string nome;
      ...
   public :    ...
};
```

```
class PessoaFisica : public Pessoa
{
   private : string CPF;
      ...
   public :   ...
};
```

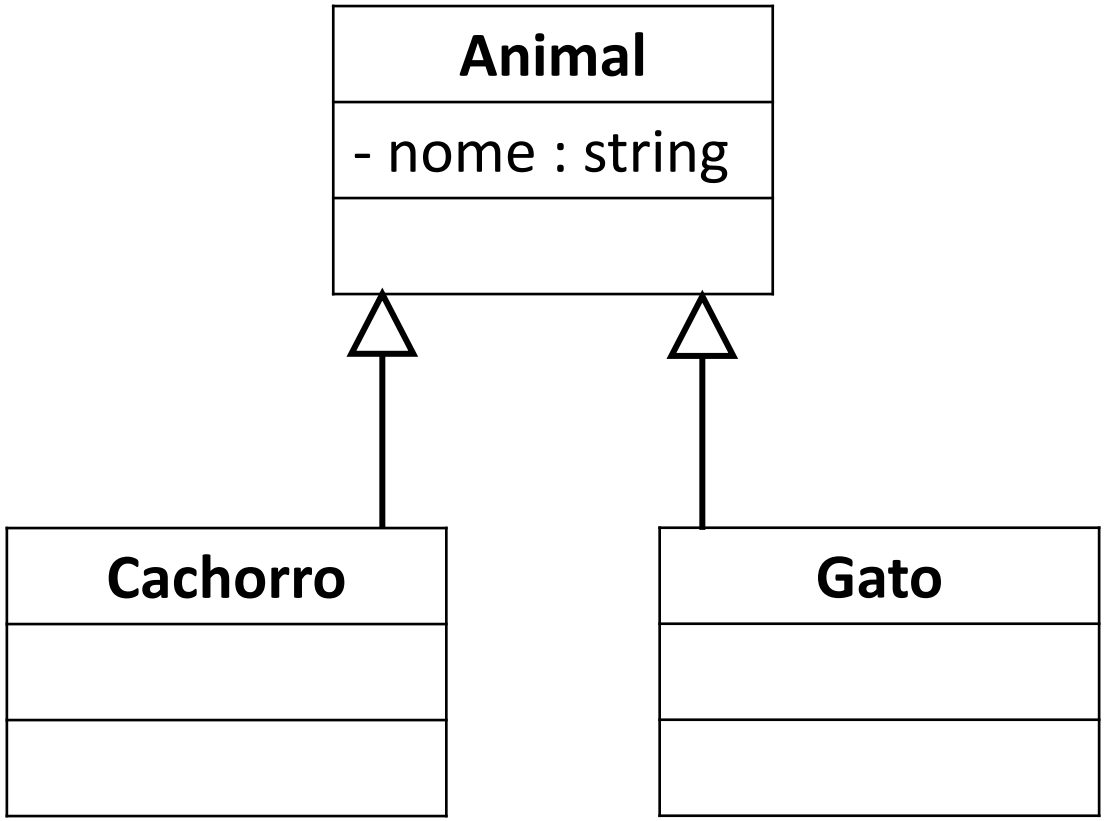
```
class PessoaJuridica : public Pessoa
{
   private :  string CNPJ;
      ...
   public :    ...
};
```

Generalização

Relacionamento do tipo: ***is a*** *(é um)*

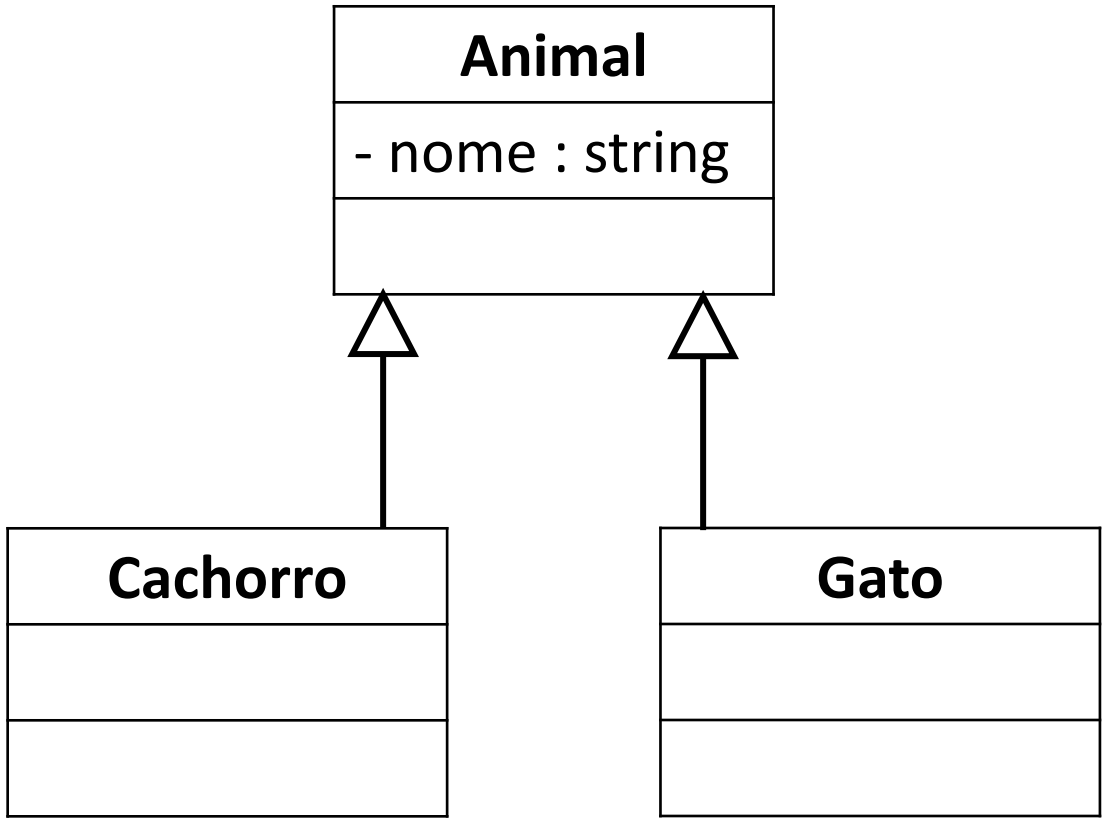
Classe-base (Superclasse) generaliza propriedades comuns (ou pura abstração)

Classes-derivadas (Subclasses) representam suas especializações

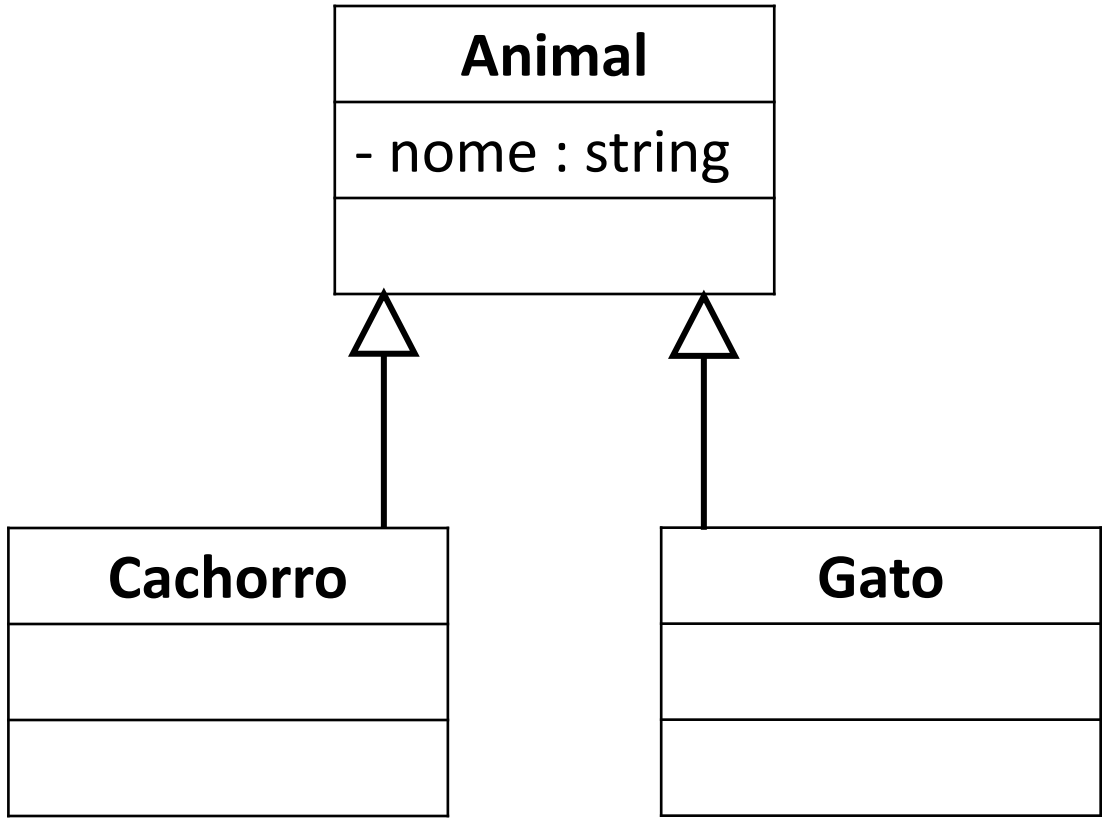


Generalização

Implemente a generalização expressa nas três classes abaixo:

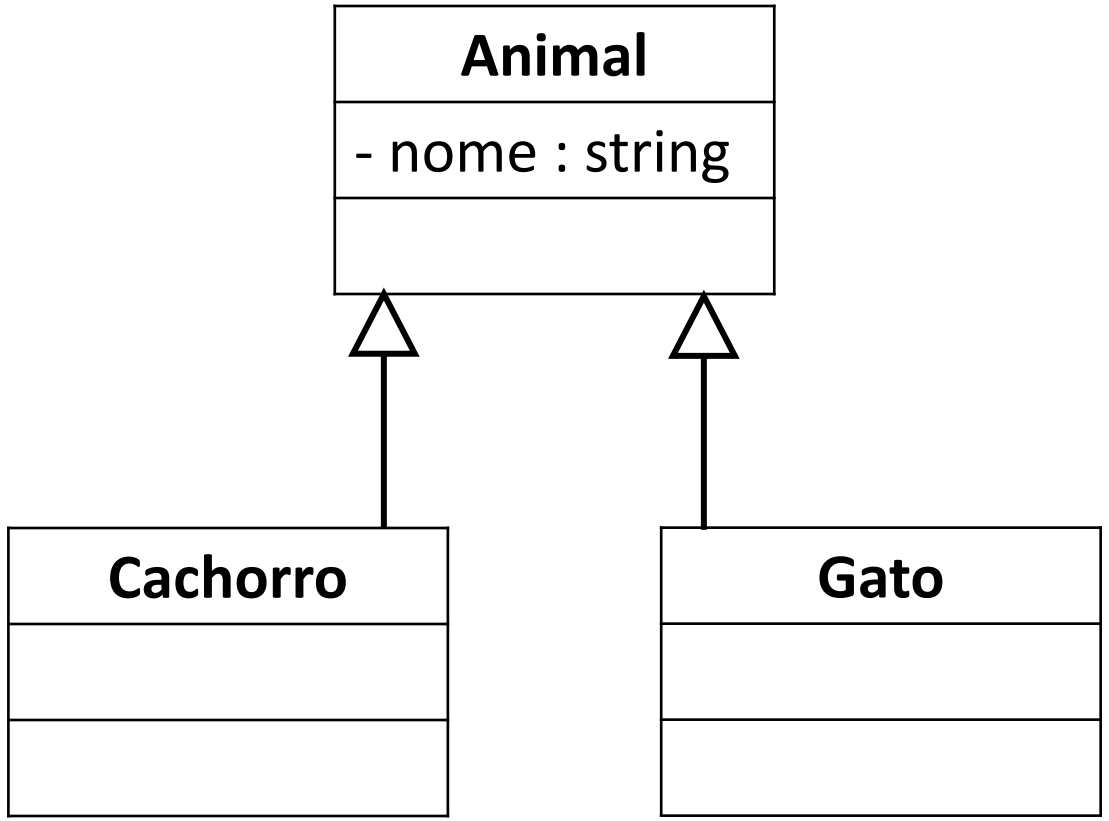


Generalização




```
class Animal
{
   private :
        string nome;
   public :

};
```

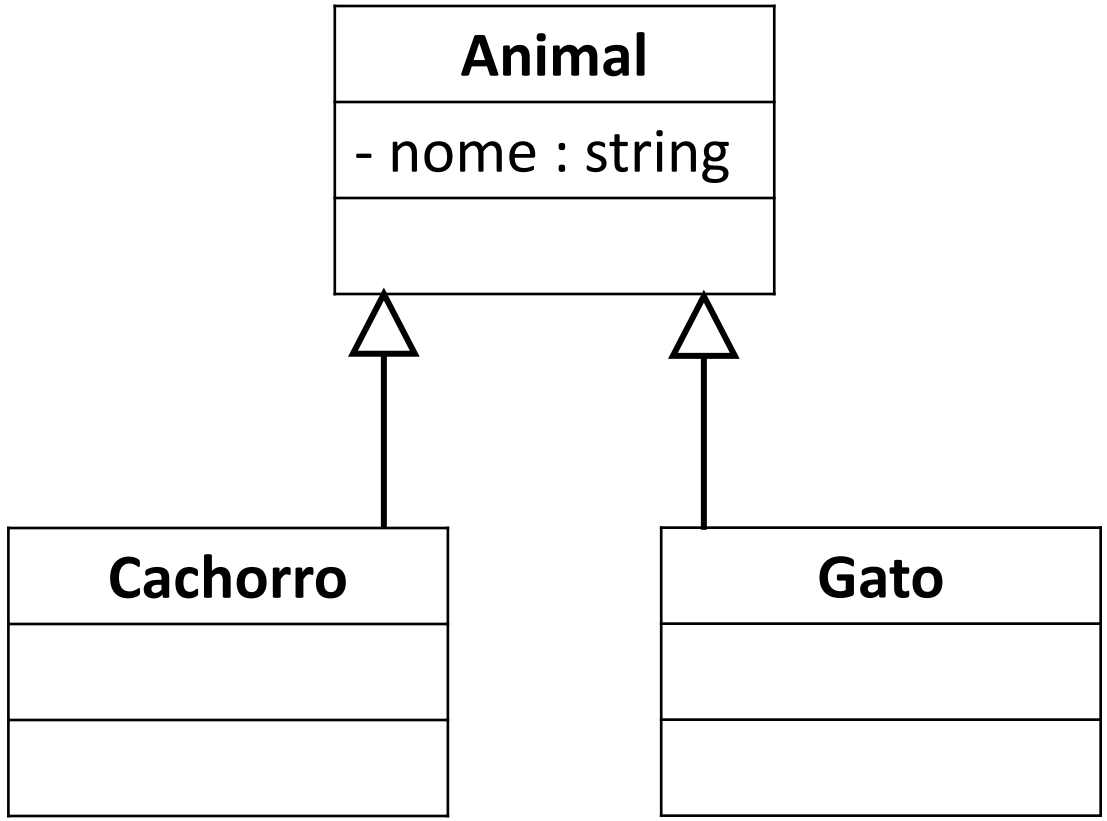
Generalização




```
class Animal
{
   private :
        string nome;
   public :

};
```

```
class Cachorro : Animal
{
   public :

};
```

Generalização

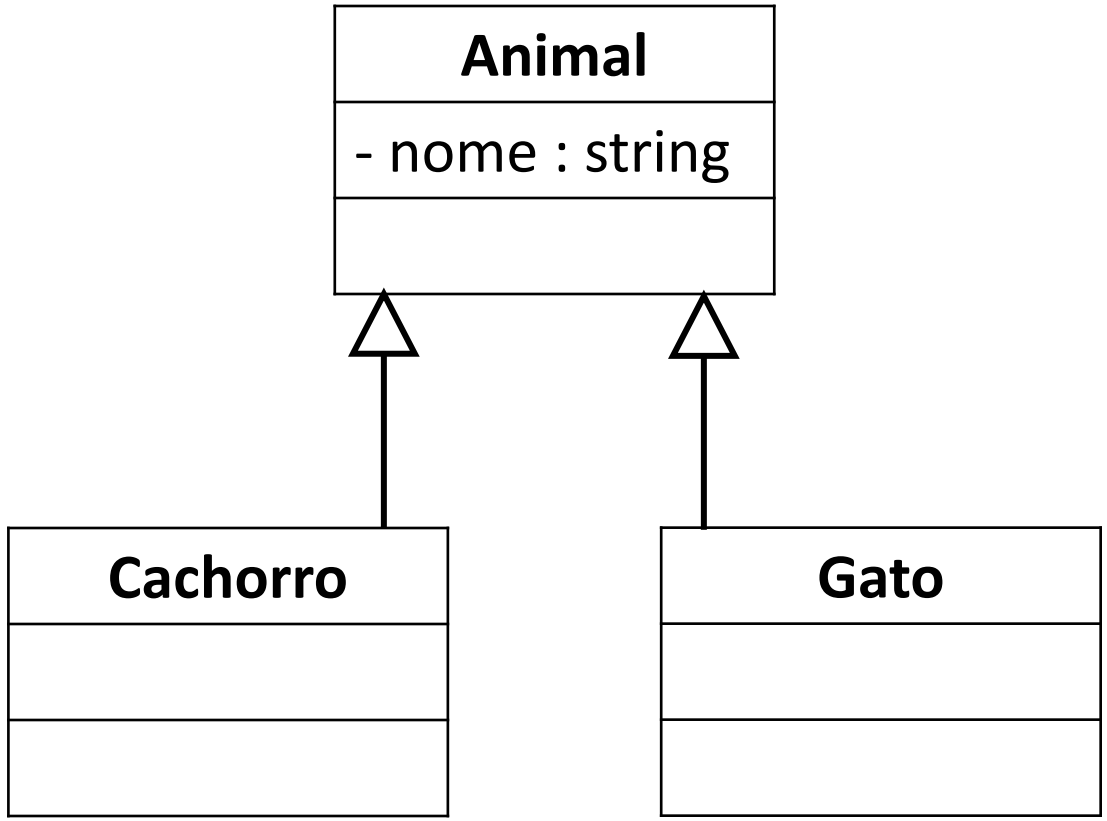



```
class Animal
{
   private :
        string nome;
   public :

};
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```

Generalização



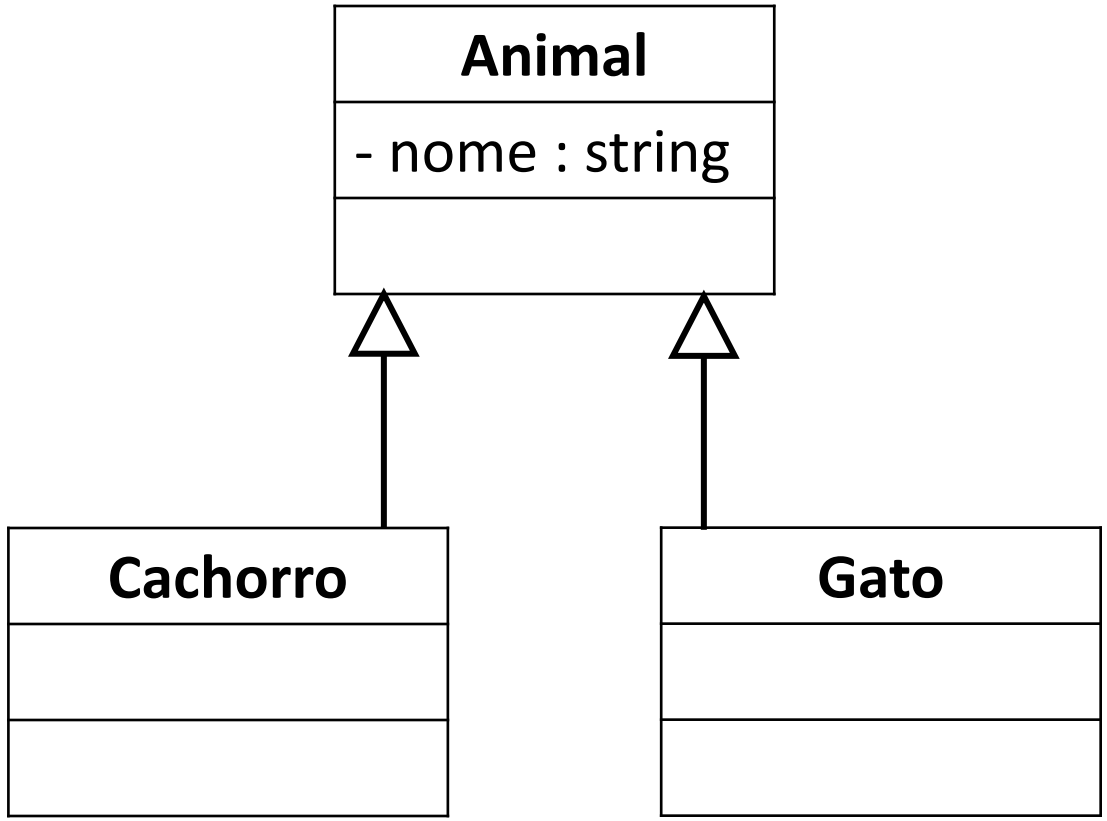

```
class Animal
{
    private :
        string nome;
    public :

};
```

```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...
```

```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```

Generalização

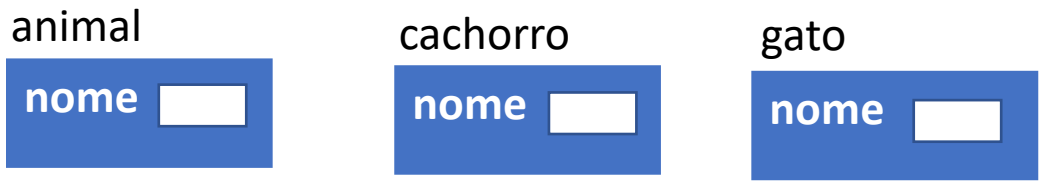



```
class Animal
{
    private :
        string nome;
    public :

};
```

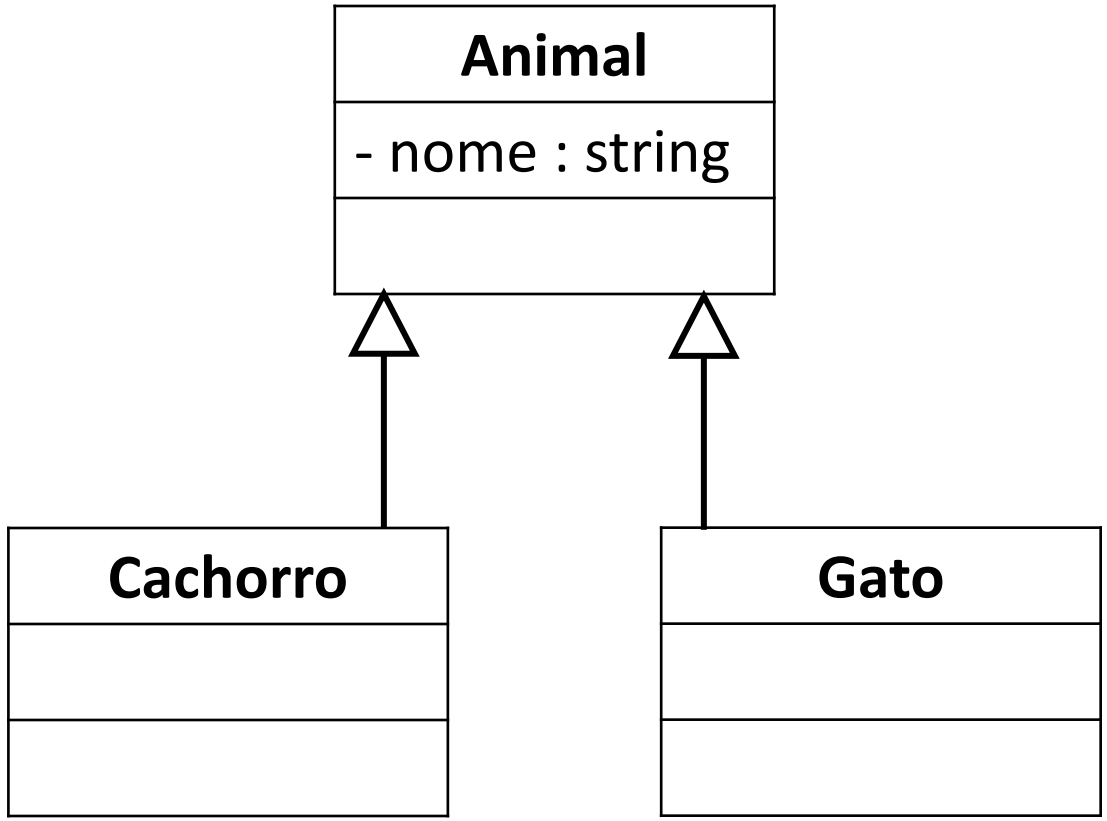
```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...
```

```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```



Generalização

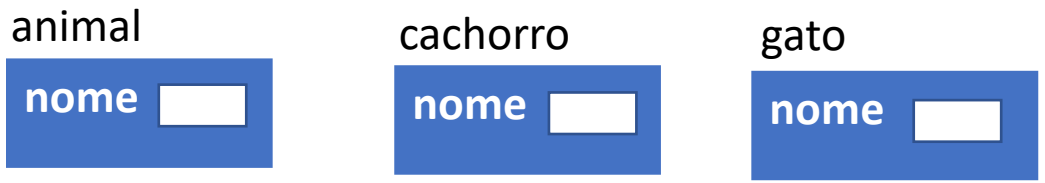



```
class Animal
{
    private :
        string nome;
    public :

};
```

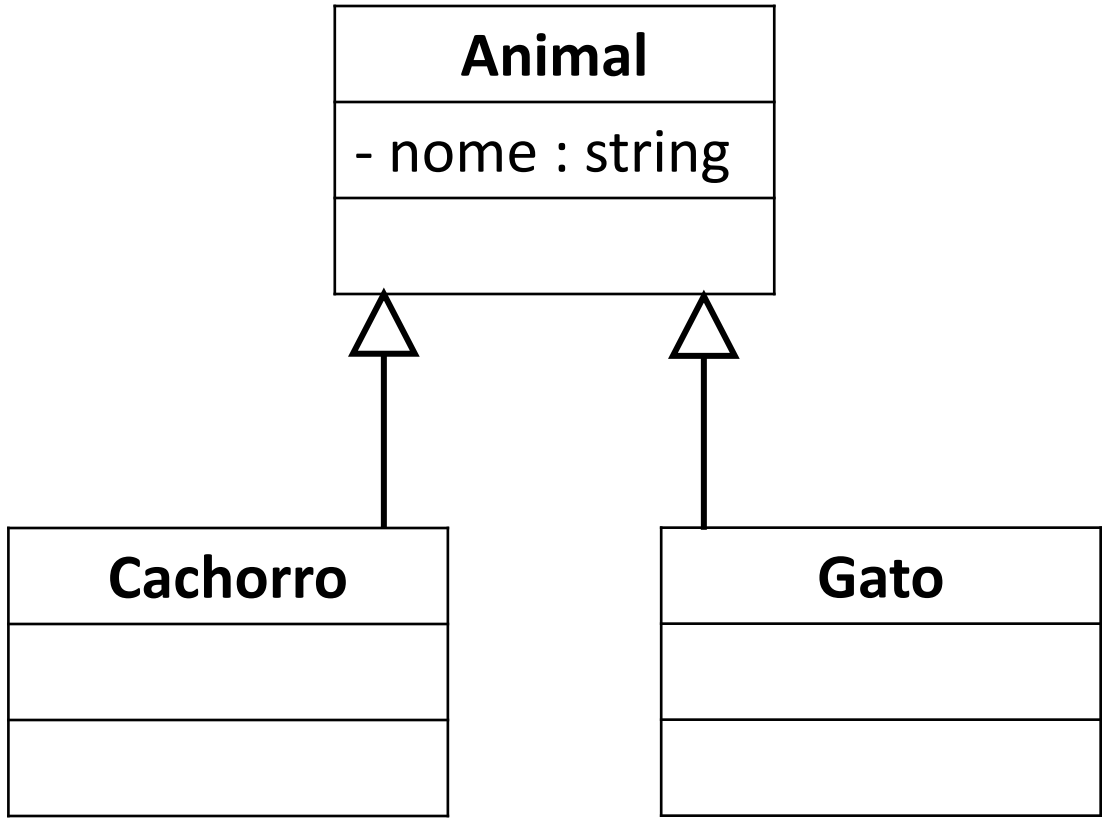
```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```

```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...

cout << animal.nome;
```



Generalização



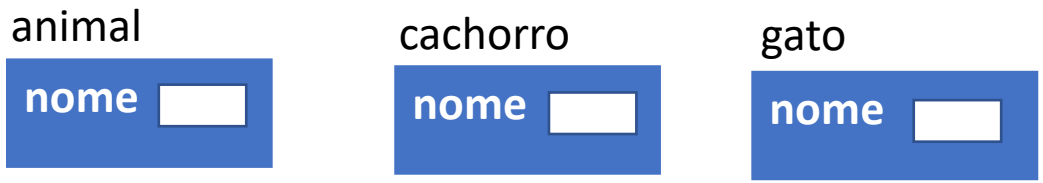

```
class Animal
{
    private :
        string nome;
    public :

};
```

```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...
```

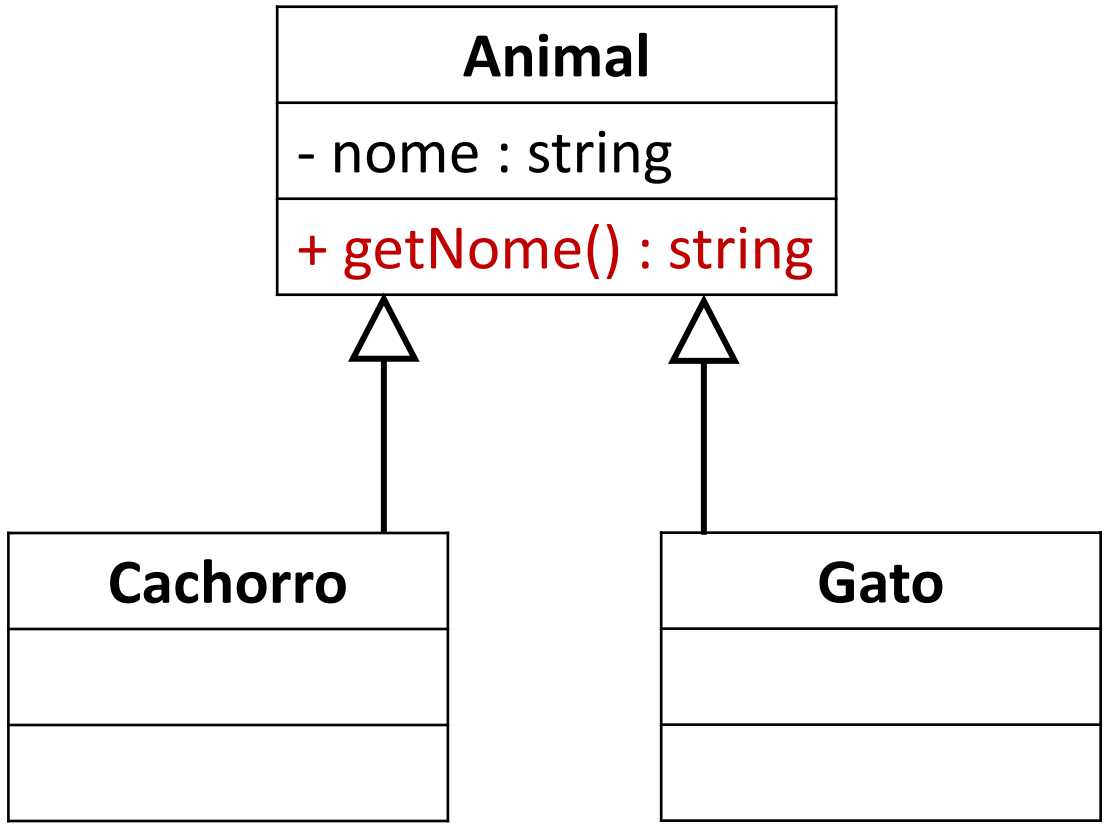
✖ `cout << animal.nome; //Privado`

```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```



Generalização

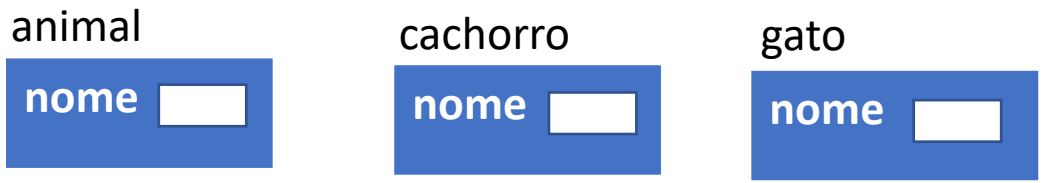



```
class Animal
{
   private :
        string nome;
   public :

};
```

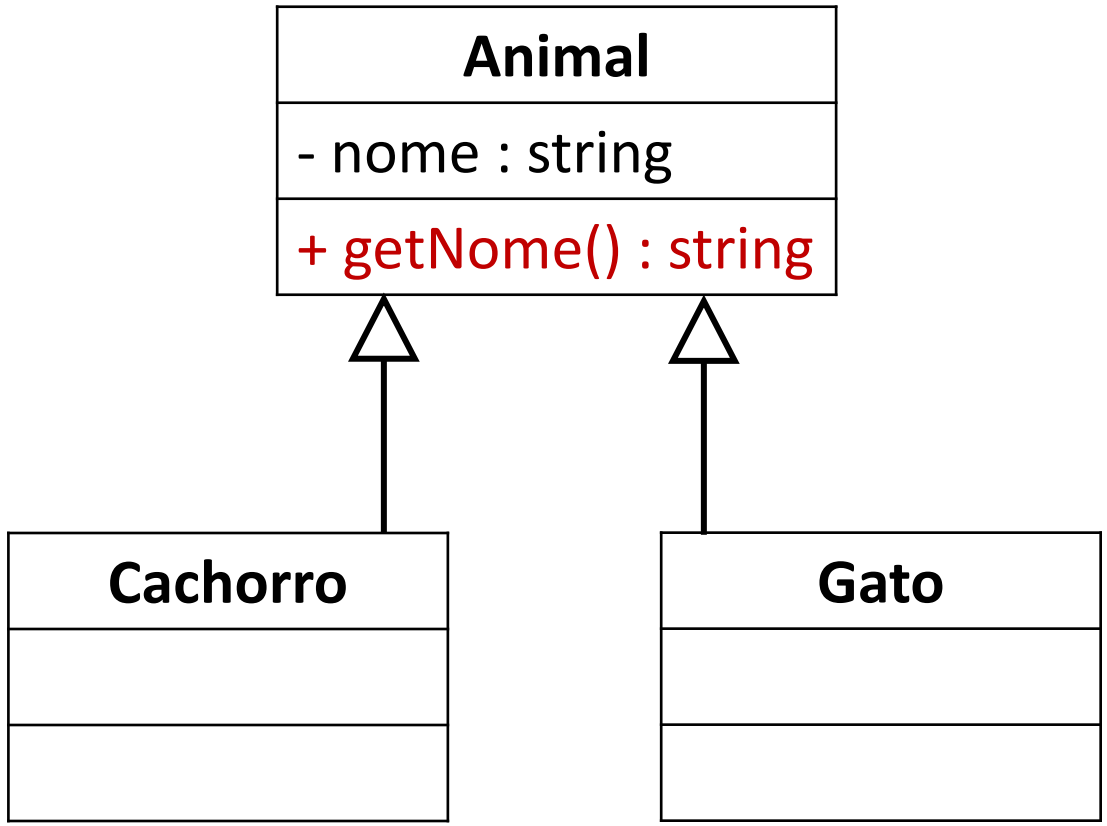
```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...

cout << animal.getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

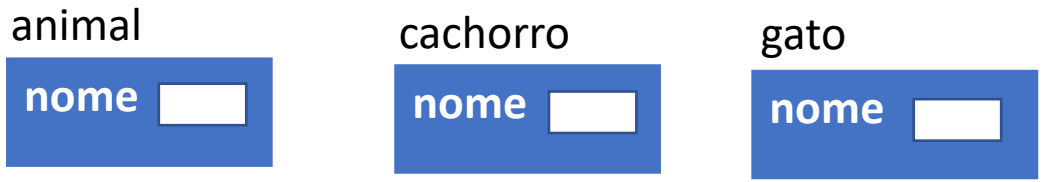



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

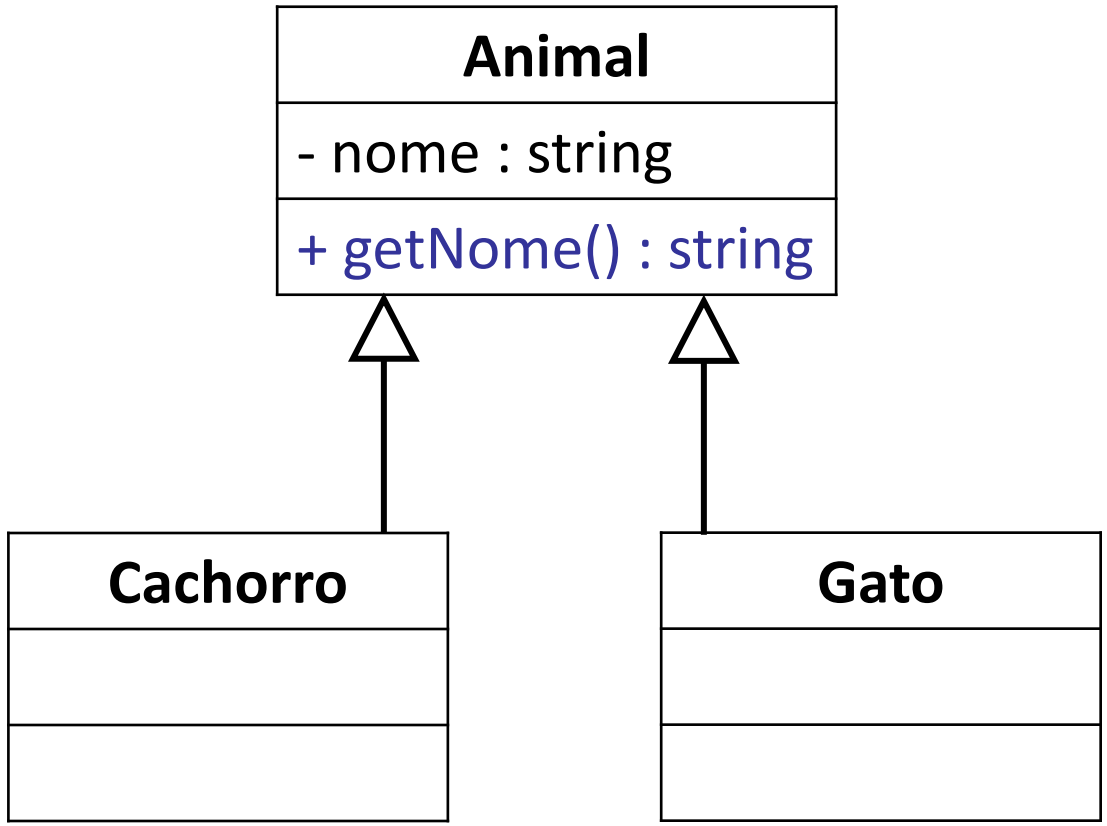
```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```

```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...

cout << animal.getNome();
```



Generalização




```
class Animal
{
   private :
       string nome;
   public :
       string getNome()
       {
           return nome;
       }
};
```

```
...
Animal* animal;
...
Cachorro* cachorro;
...
Gato* gato;
...

cout << animal.getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

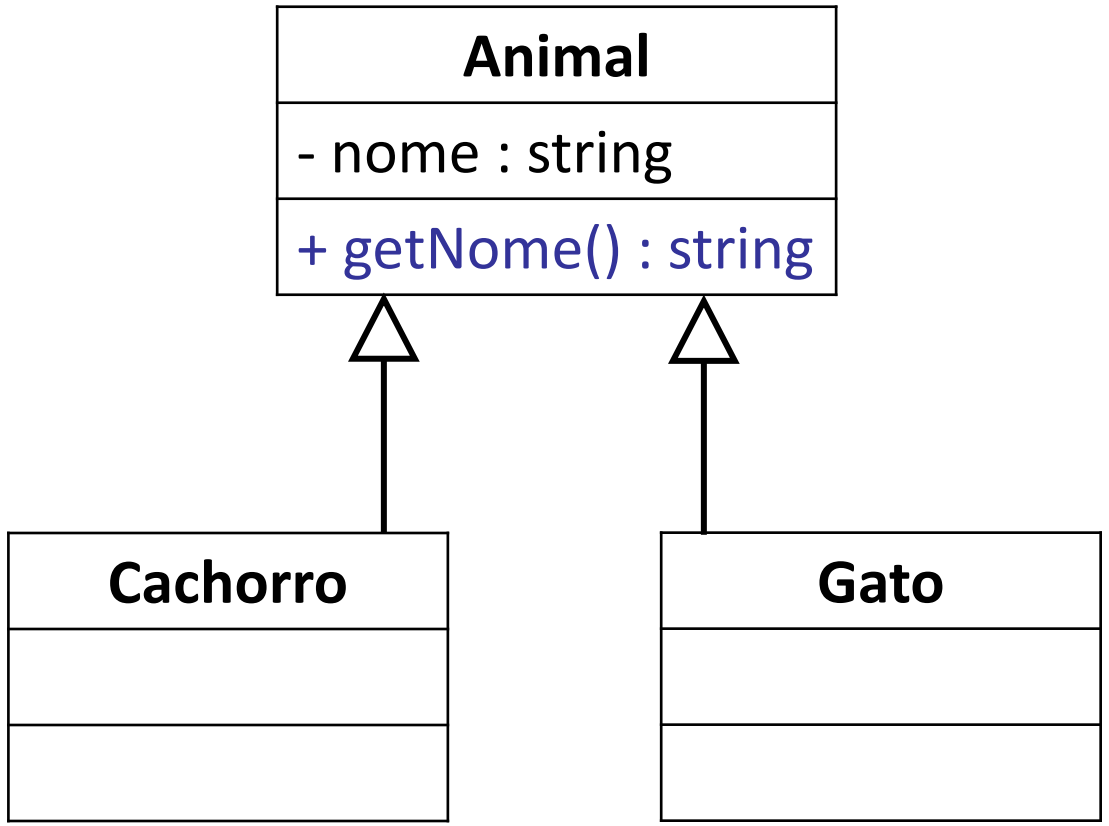
```
class Gato : Animal
{
   public :

};
```

animal

cachorro

gato

Generalização




```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

```
...
Animal* animal;
...
Cachorro * cachorro;
...
Gato * gato;
...
```

✖ `cout << animal.getNome();`

```
class Cachorro : Animal
{
   public :

};
```

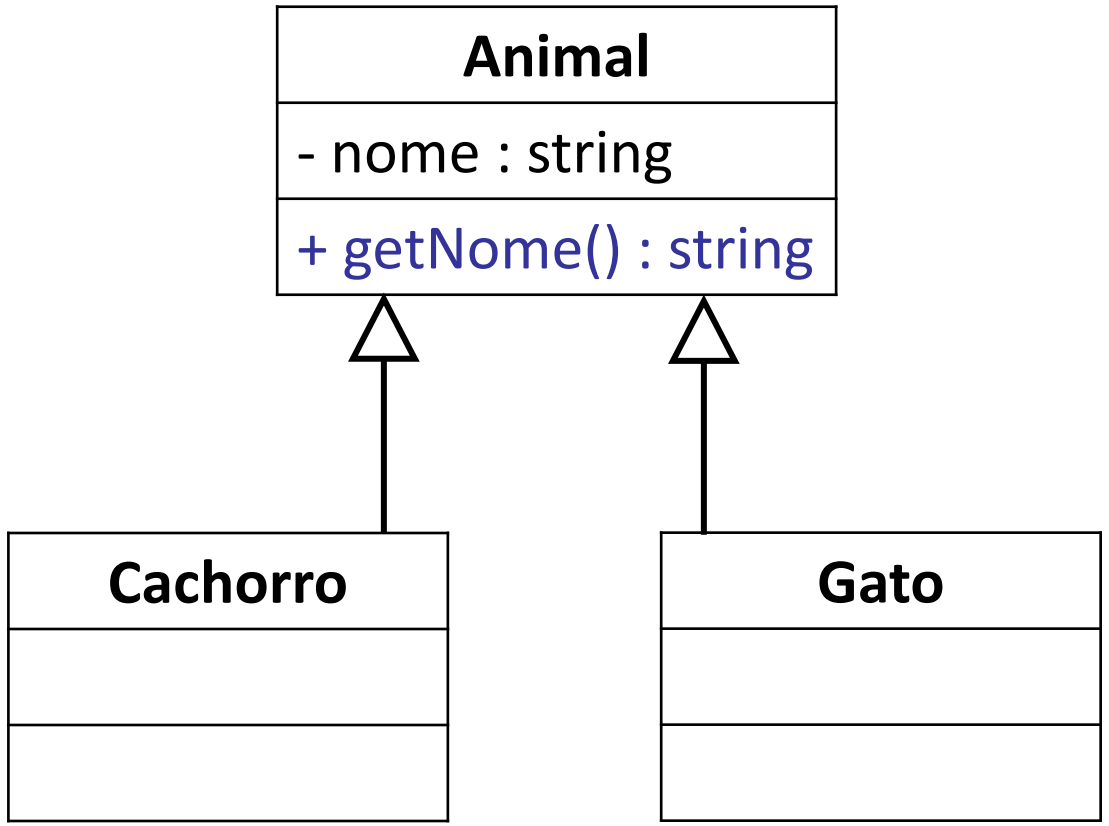
```
class Gato : Animal
{
   public :

};
```

animal

cachorro

gato

Generalização




```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

```
...
Animal* animal;
...
Cachorro * cachorro;
...
Gato * gato;
...
```

✖ `cout << animal->getNome();`

```
class Cachorro : Animal
{
   public :

};
```

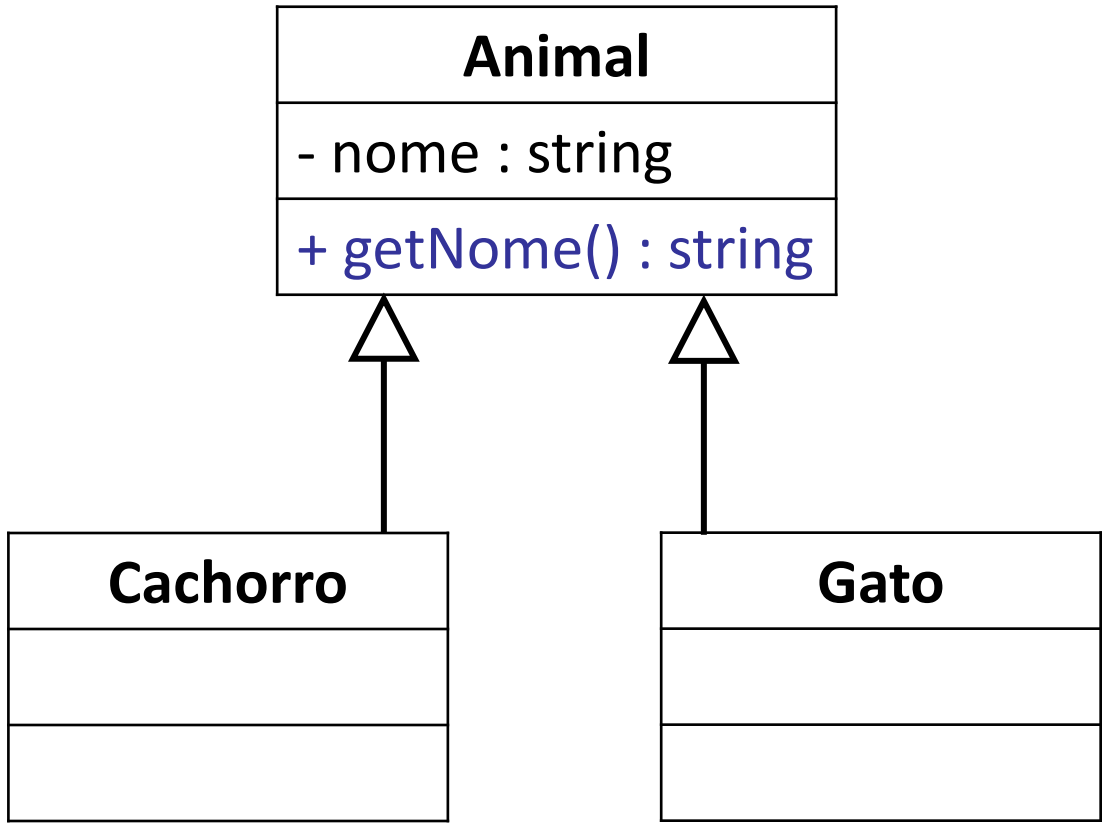
```
class Gato : Animal
{
   public :

};
```

animal

cachorro

gato

Generalização

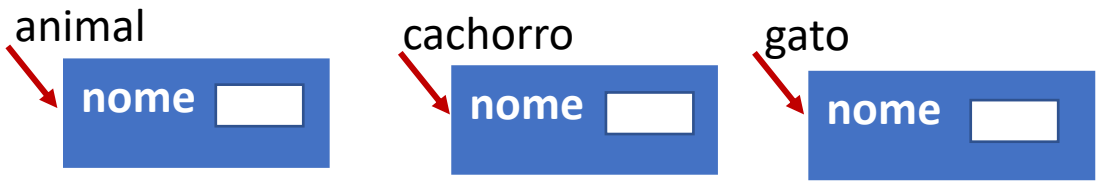



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

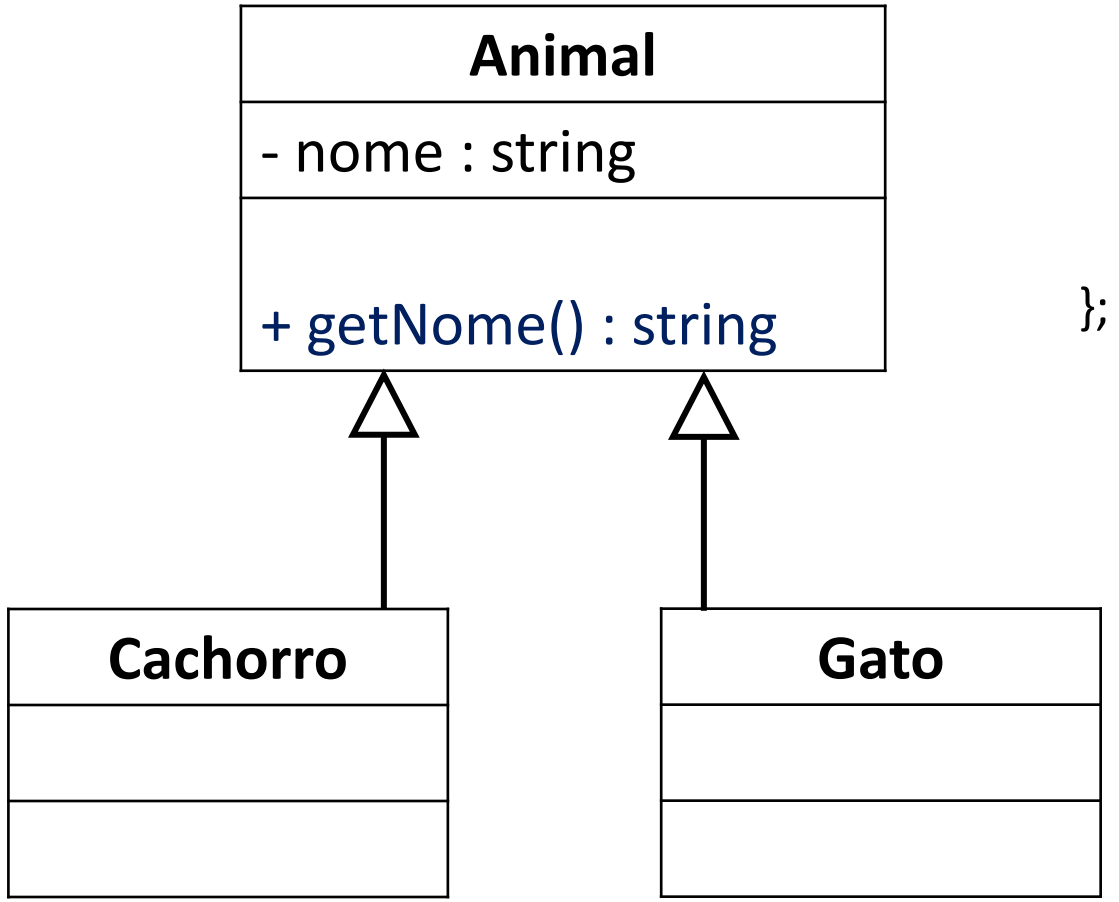
```
...
Animal* animal;
animal = new Animal;
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

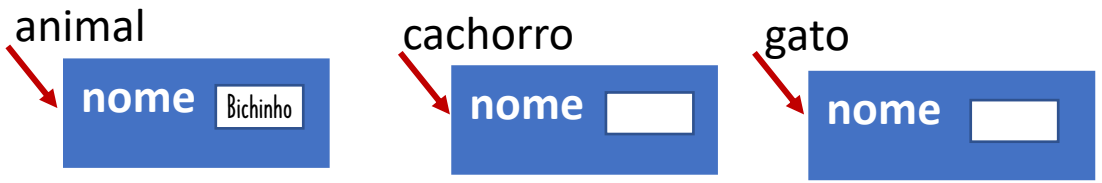



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

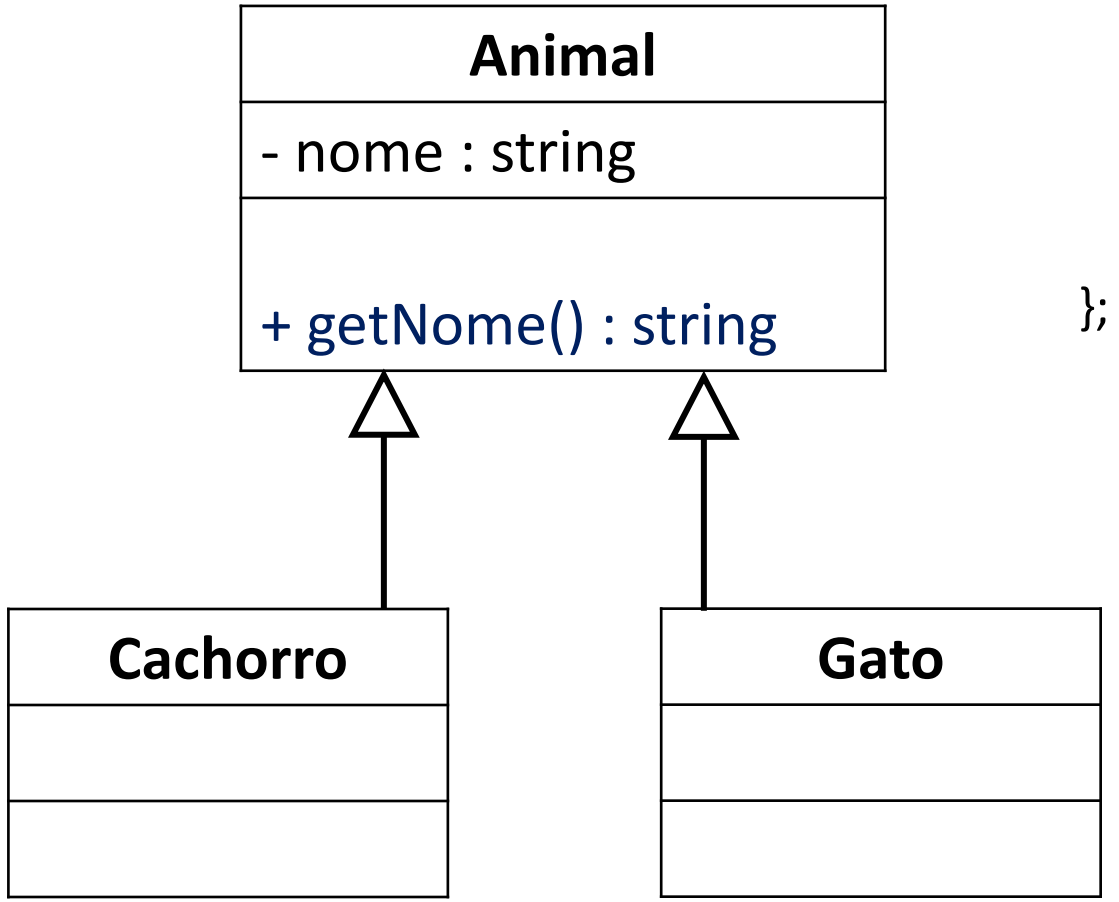
```
...
Animal* animal;
animal = new Animal("Bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

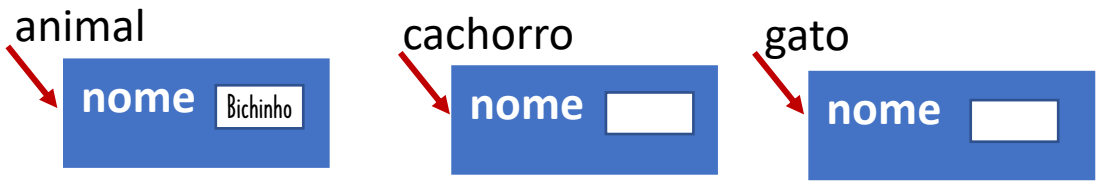



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

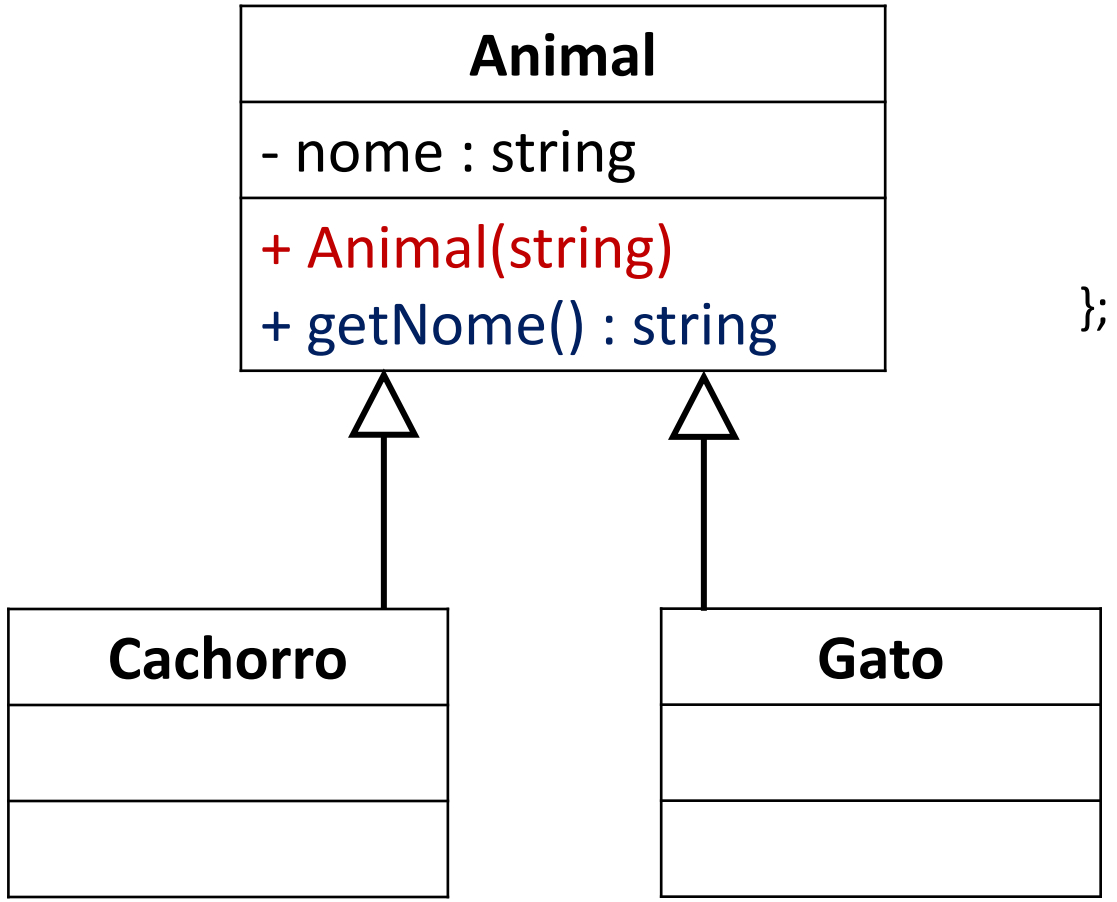
```
...
Animal* animal;
animal = new Animal("Bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

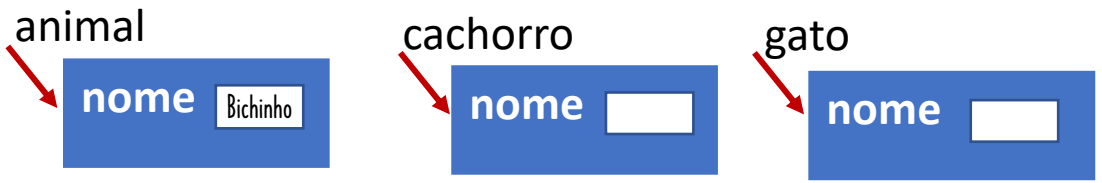



```
class Animal
{
   private :
      string nome;
   public :
      string getNome()
      {
         return nome;
      }
};
```

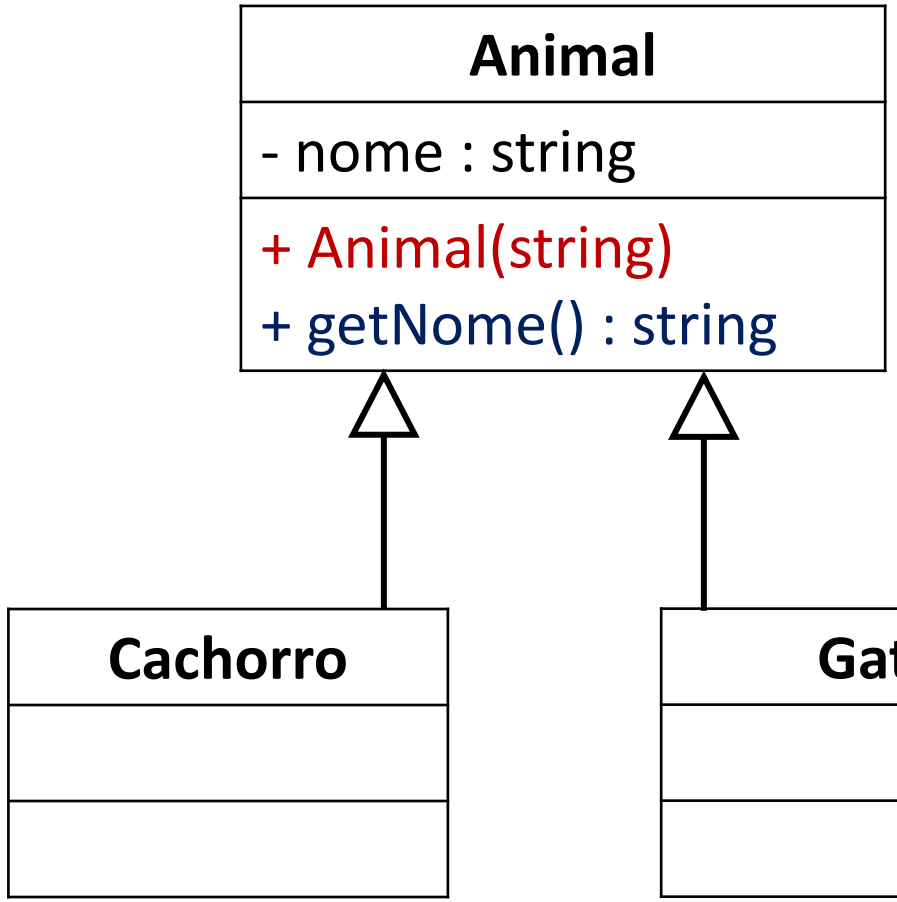
```
...
Animal* animal;
animal = new Animal("Bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização




```
class Animal
{
   private :
        string nome;
   public :
        Animal(string nome)
        {
             this->nome= nome;
        }
        string getNome()
        {
            return nome;
        }
};
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```

```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```