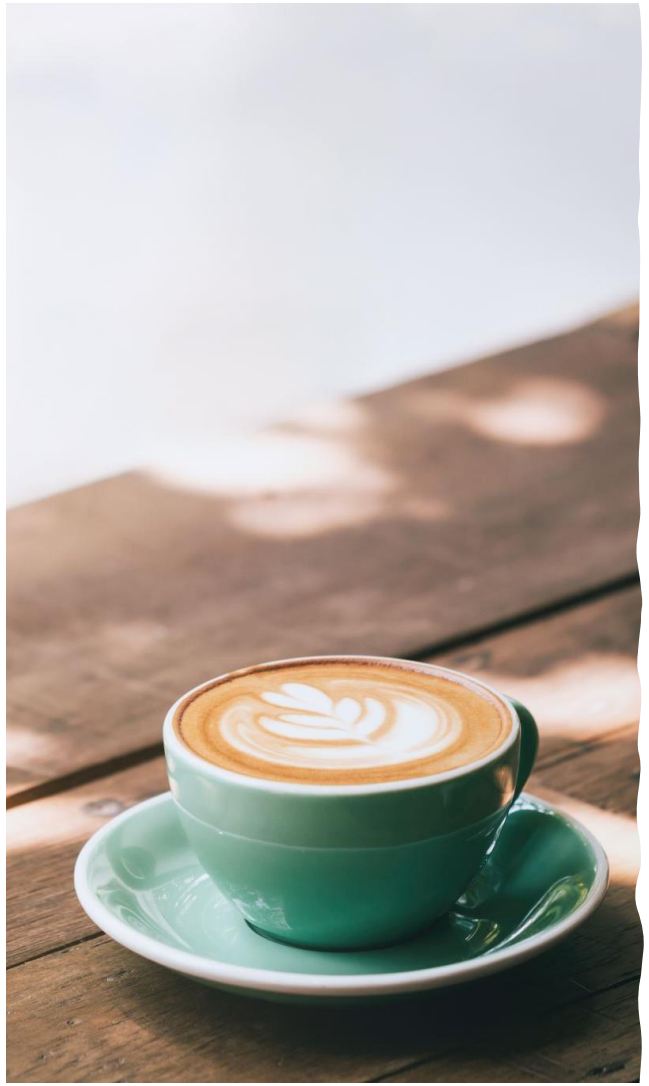


Uma introdução à Orientação por Objetos

Generalização (herança)

Maratona de Programação
28 e 29/nov

Necessário ser realizada presencialmente, no exato horário do grupo matriculado:

G1: Quinta, 7h

G2: Quinta, 8h50

G3: Sexta, 7h

G4: Sexta, 8h50

Hoje: Bênção da Sede do *Campus* Lourdes

Sede do *Campus* Lourdes
Rua Alvarenga Peixoto, 159

Grão Chanceler da PUC Minas, Dom Walmor
Magnífico Reitor da PUC Minas, Pe. Luís Henrique

Aula anterior: *introdução à generalização (herança)*

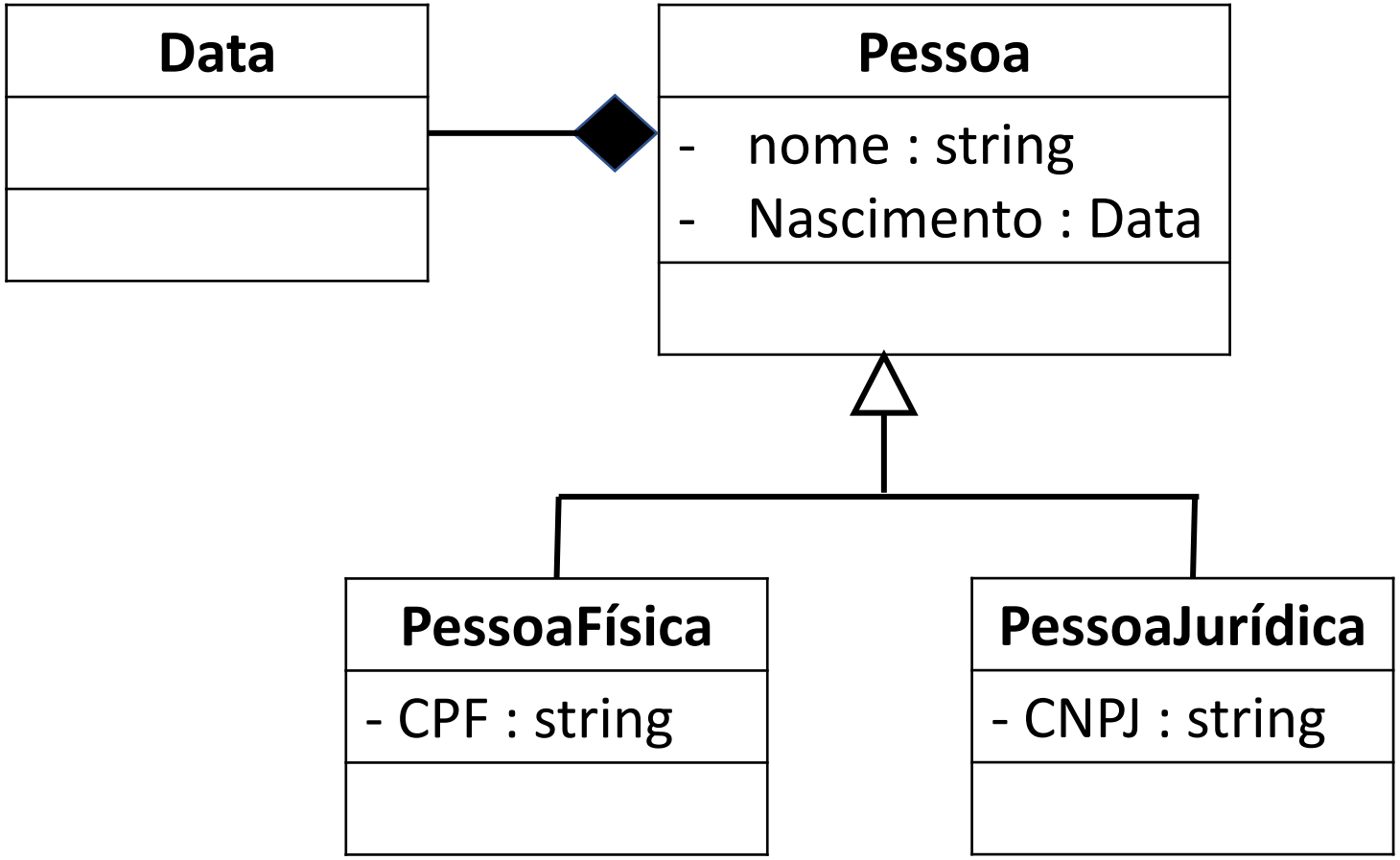
Generalização (herança)

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

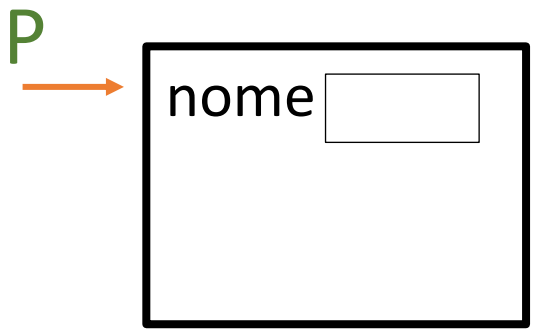
Uma classe (base) pode se especializar em outras (derivadas)

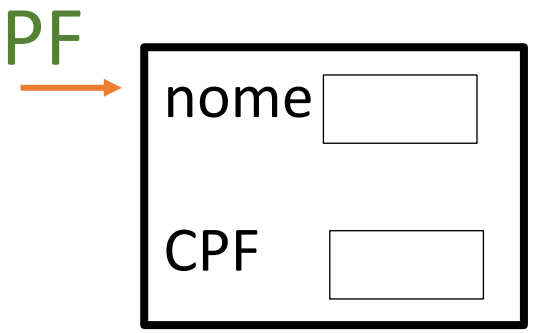
Uma classe (derivada) pode herdar as propriedades descritas em outra (base)

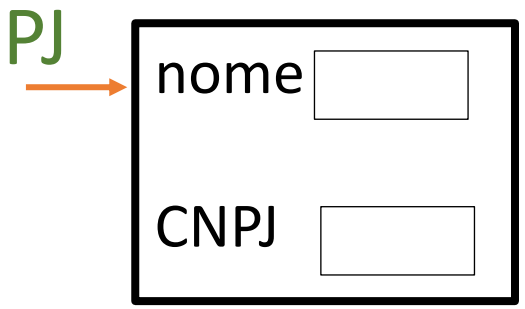
Relacionamento do tipo: ***is a***



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







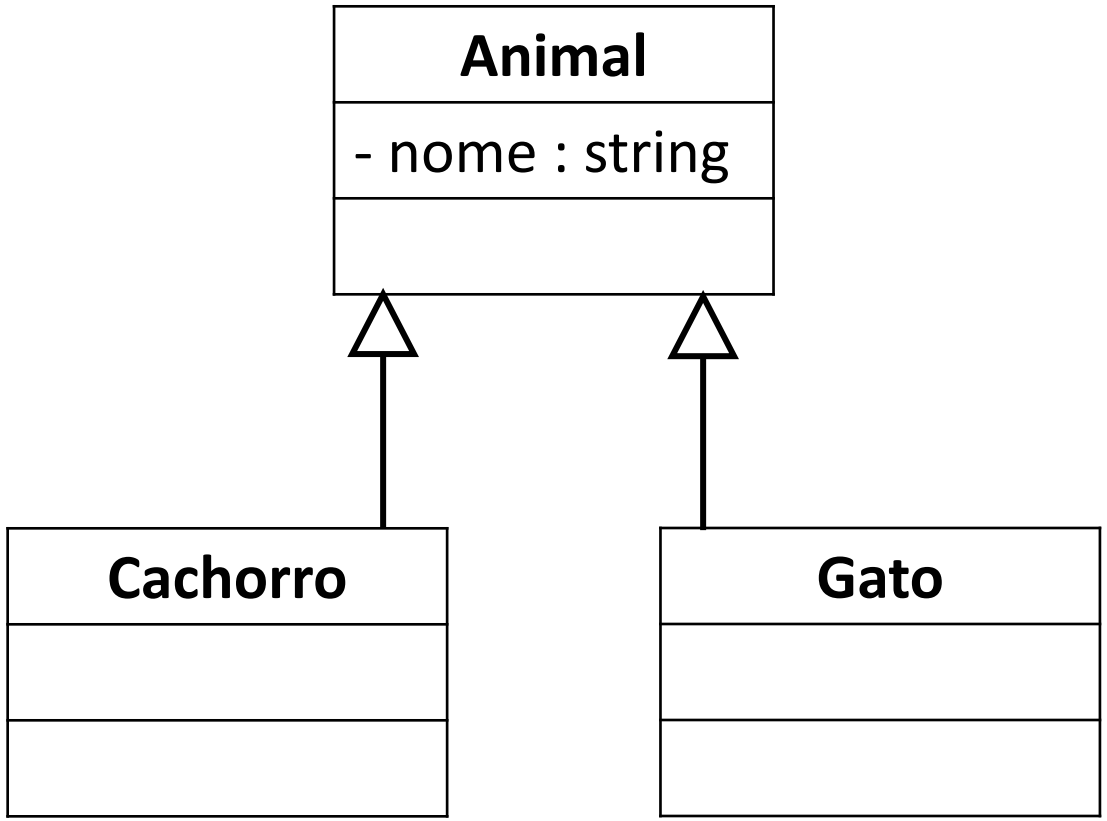

```
class Pessoa
{
   private : string nome;
      ...
   public :    ...
};
```

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

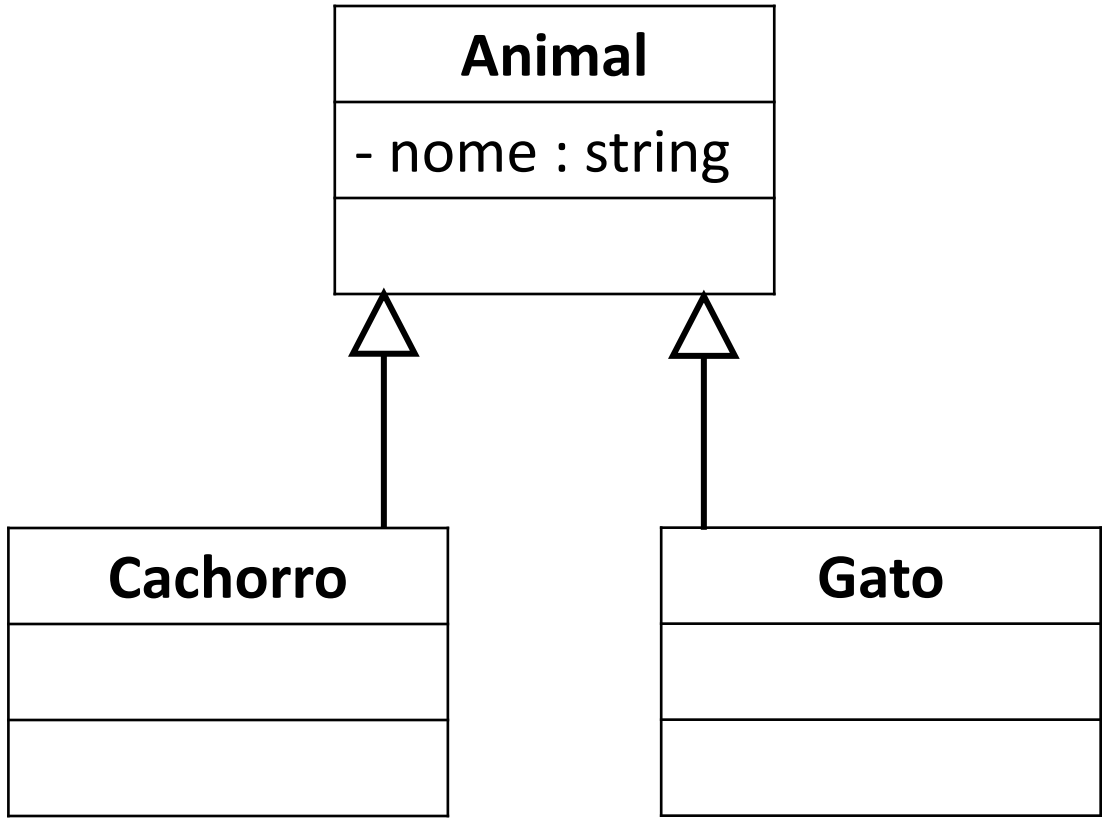
```
class PessoaJuridica : public Pessoa
{
   private :  string CNPJ;
      ...
   public :    ...
};
```

Generalização

Implemente a generalização expressa nas três classes abaixo:

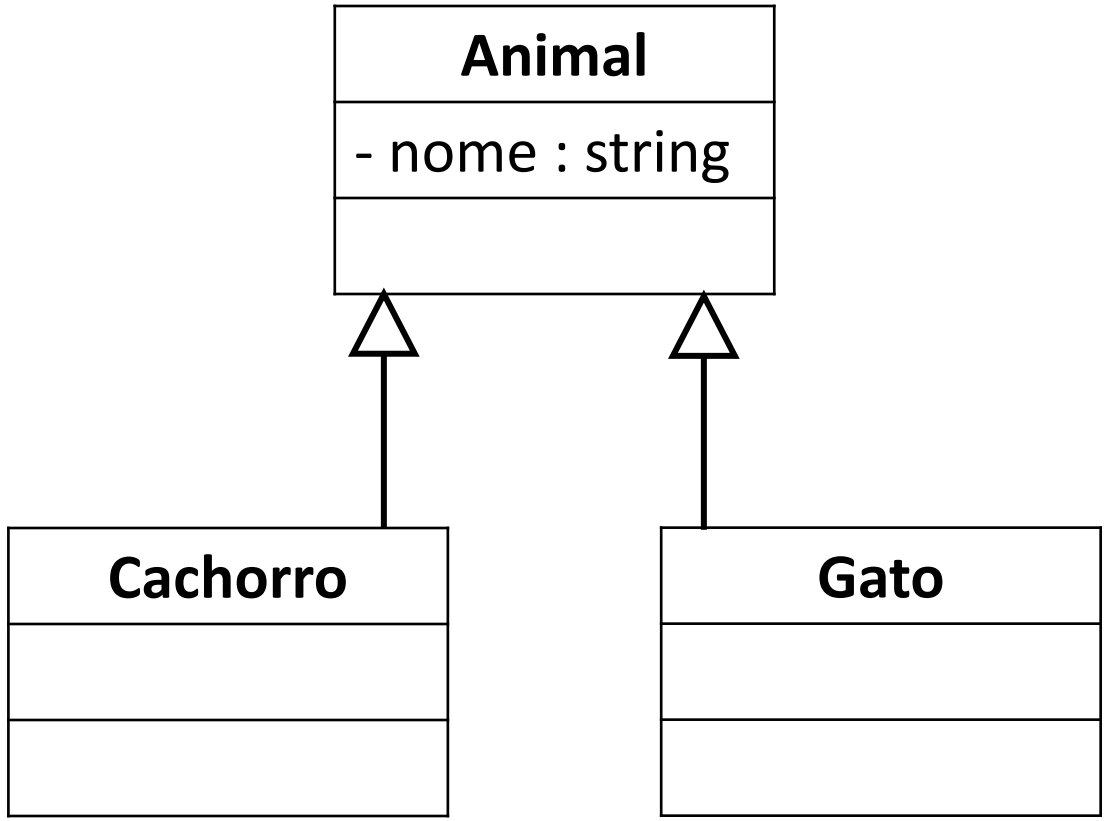


Generalização




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

};
```

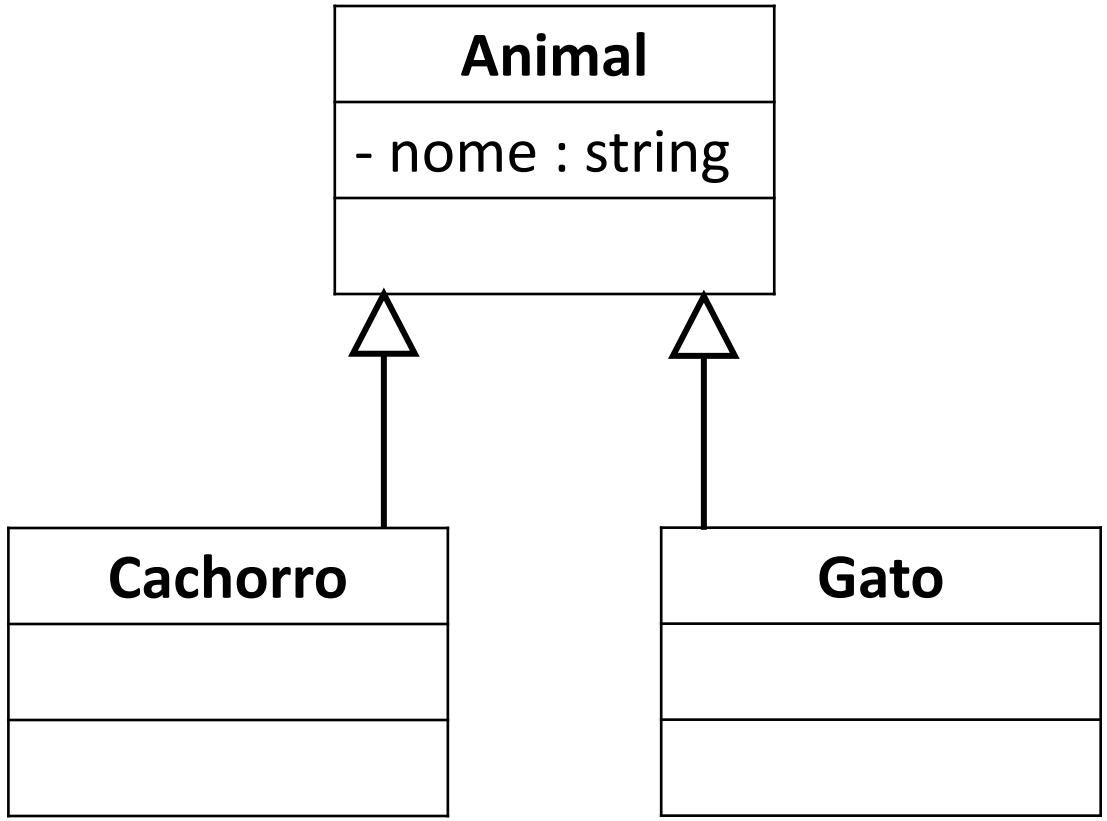
Generalização




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

};
```

```
class Cachorro : Animal
{
   public :

};
```

Generalização

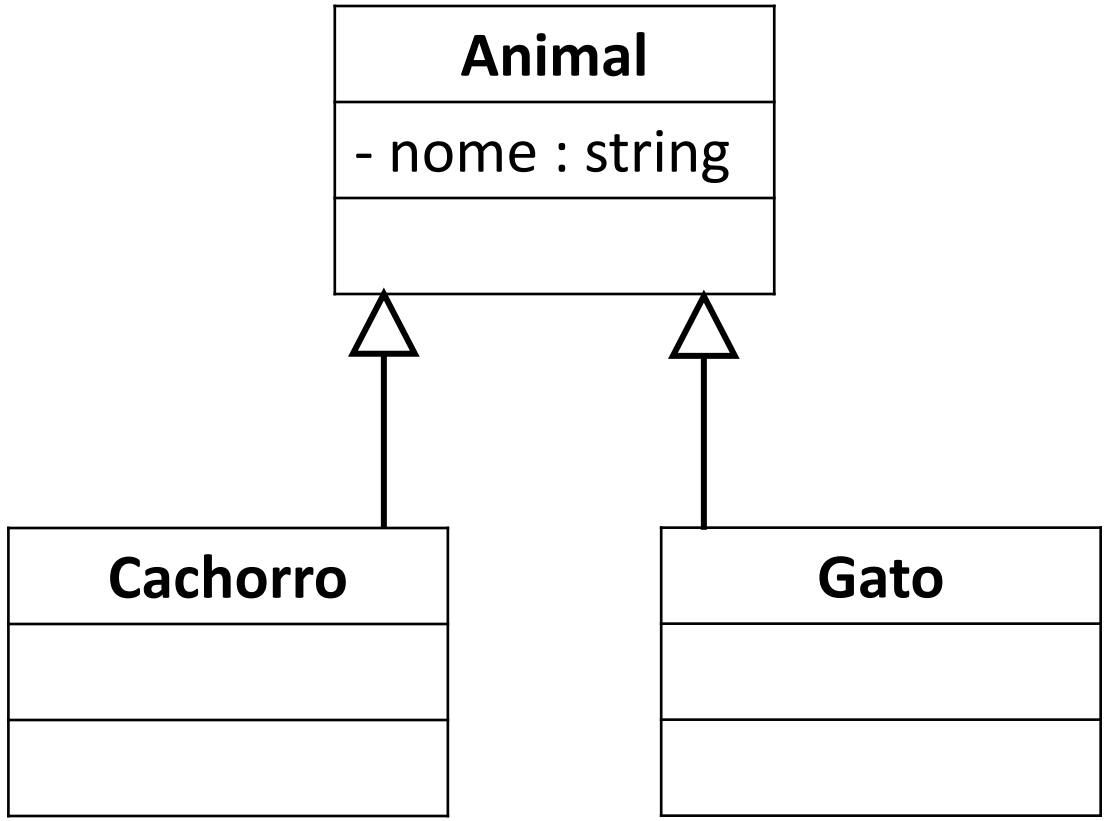



```
class Animal
{
   private :
        string nome;
   public :

};
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```

Generalização



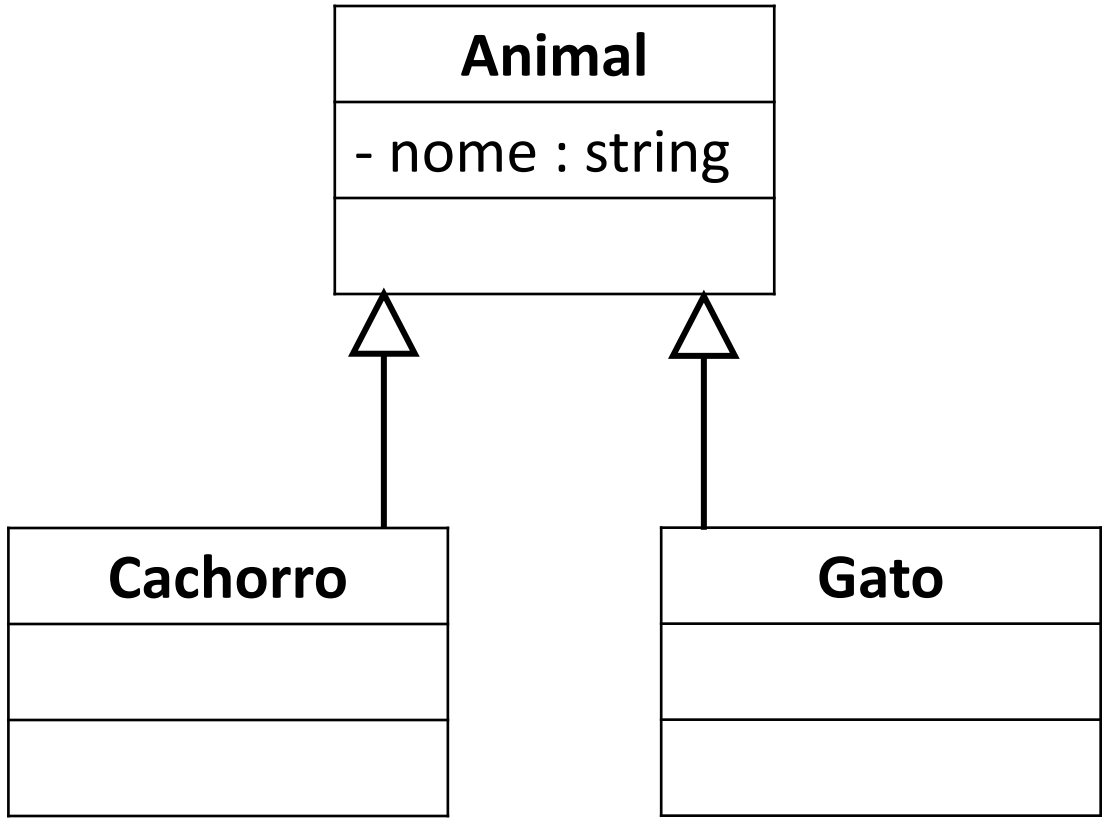

```
class Animal
{
    private :
        string nome;
    public :

};
```

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

```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```

Generalização

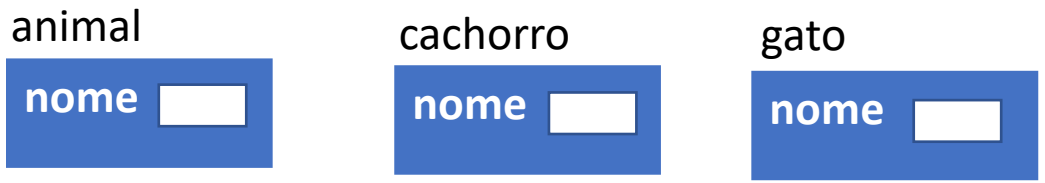



```
class Animal
{
    private :
        string nome;
    public :

};
```

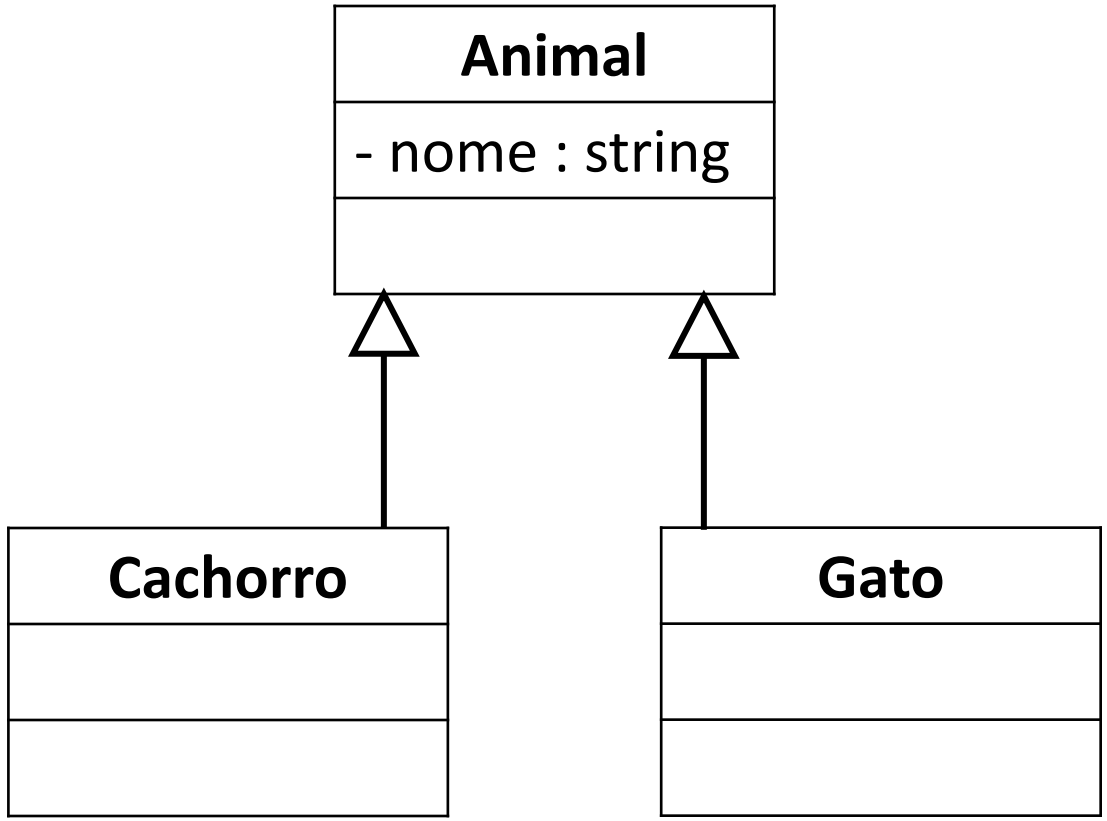
```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...
```

```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```



Generalização

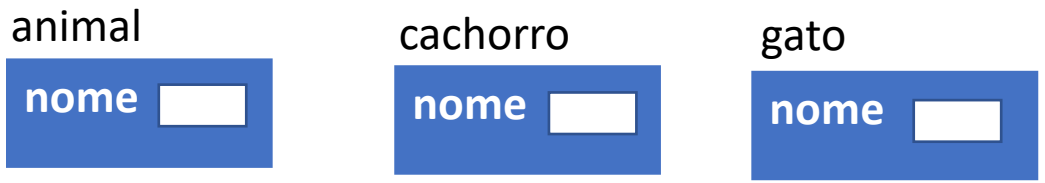



```
class Animal
{
    private :
        string nome;
    public :

};
```

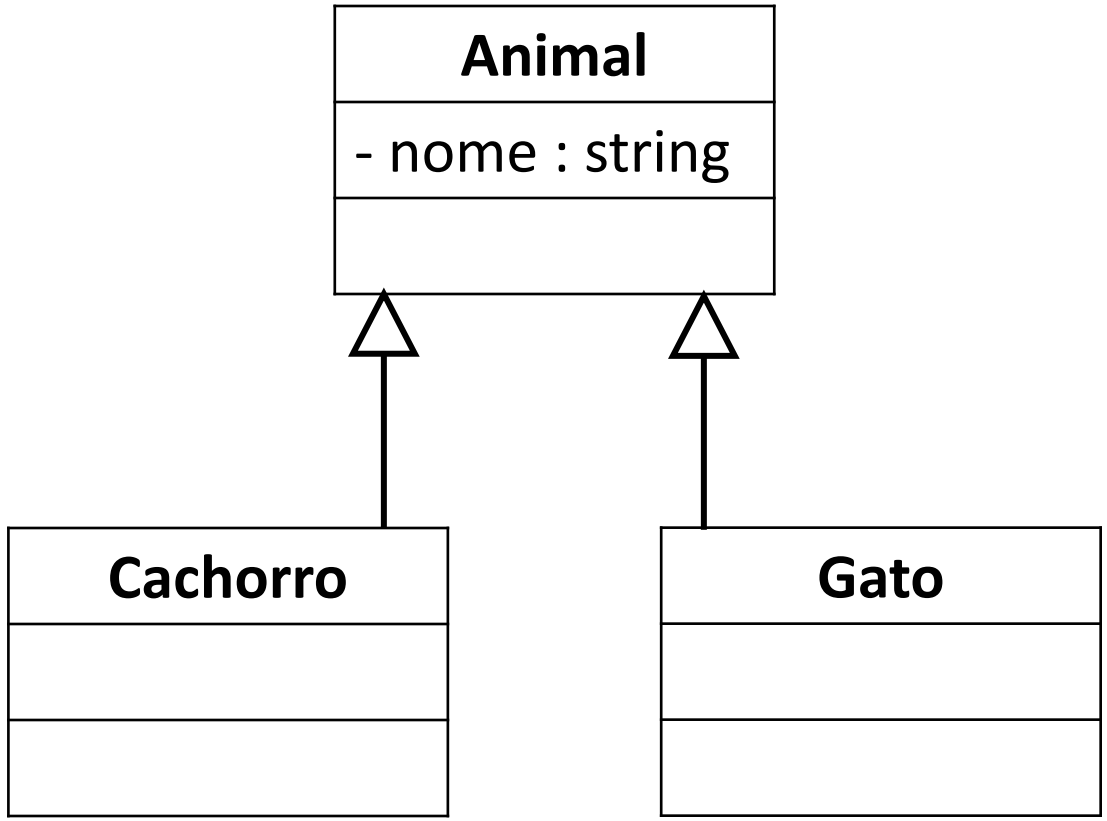
```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```

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

cout << animal.nome;
```



Generalização



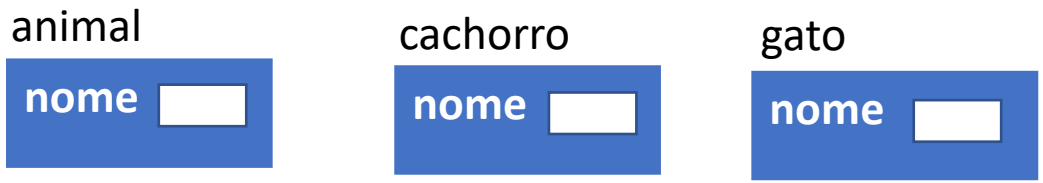

```
class Animal
{
    private :
        string nome;
    public :

};
```

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

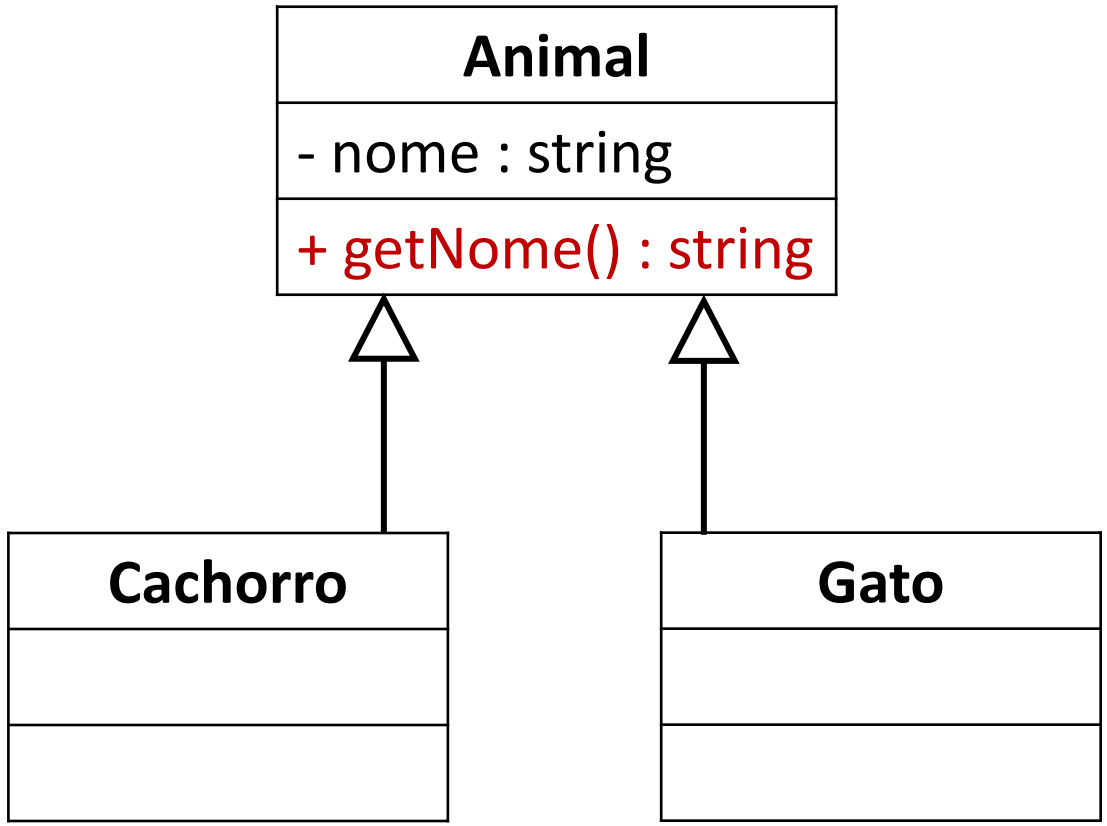
✖ `cout << animal.nome; //Privado`

```
class Cachorro : Animal
{
    public :

};
```

```
class Gato : Animal
{
    public :

};
```



Generalização

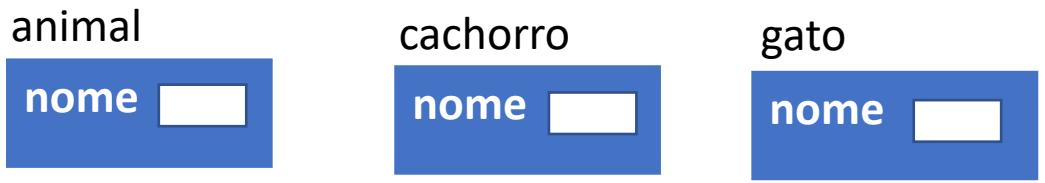



```
class Animal
{
   private :
        string nome;
   public :

};
```

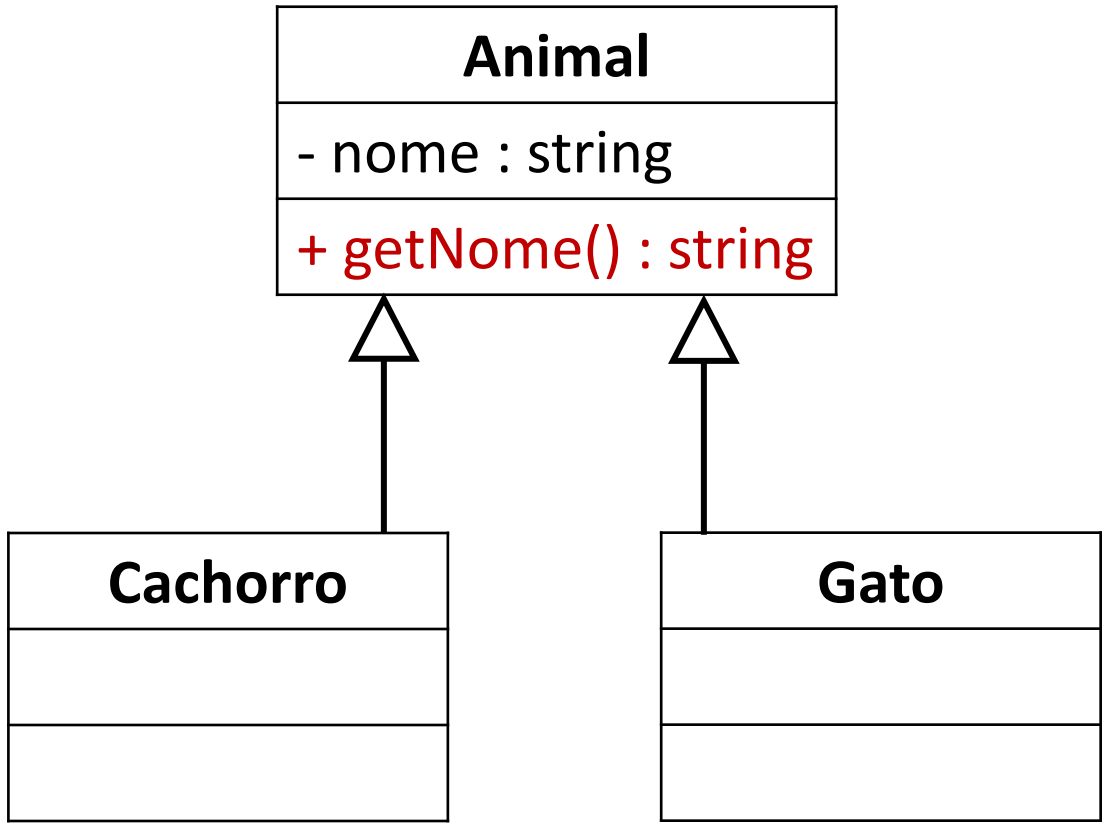
```
...
Animal animal;
...
Cachorro cachorro;
...
Gato gato;
...

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

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

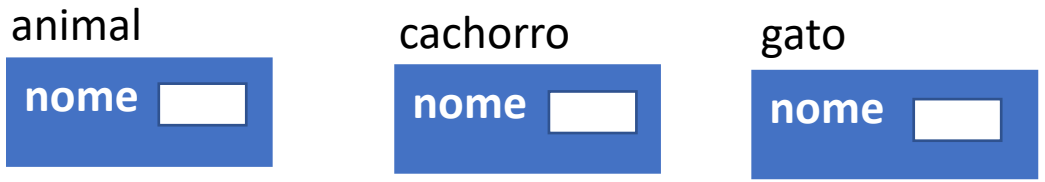



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

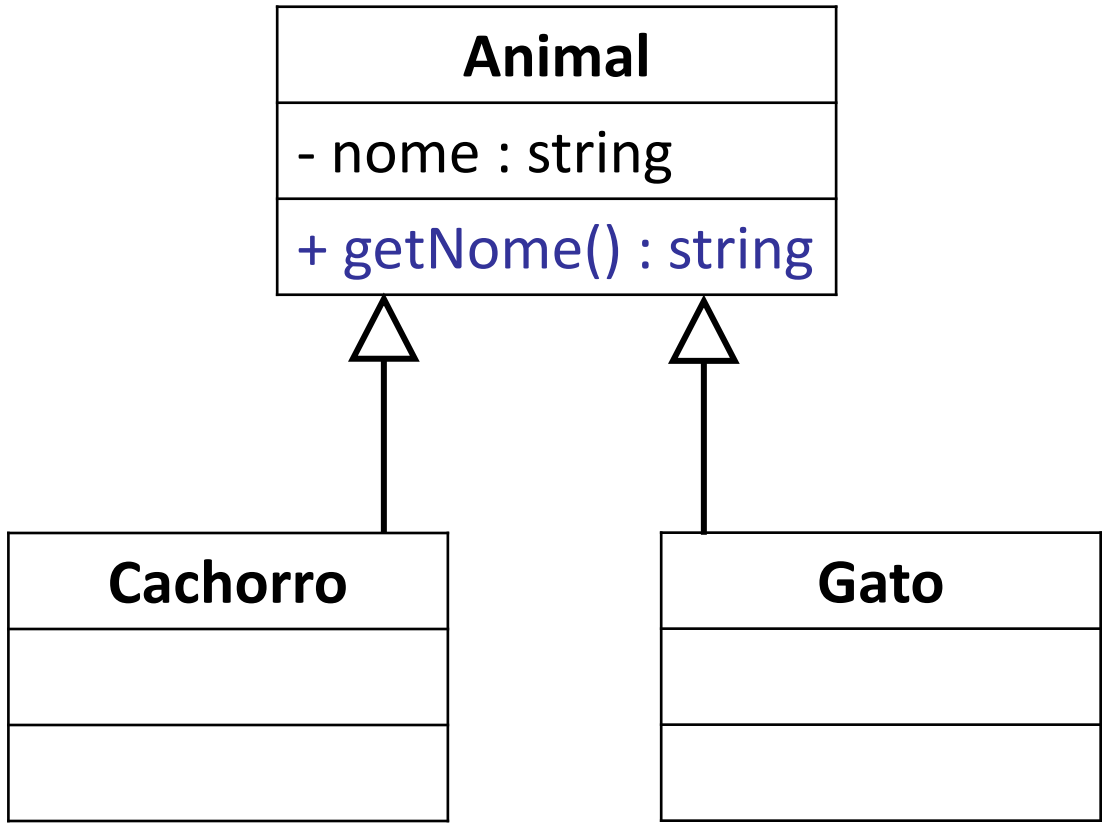
```
class Cachorro : Animal
{
   public :

};
```

```
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 :

};
```

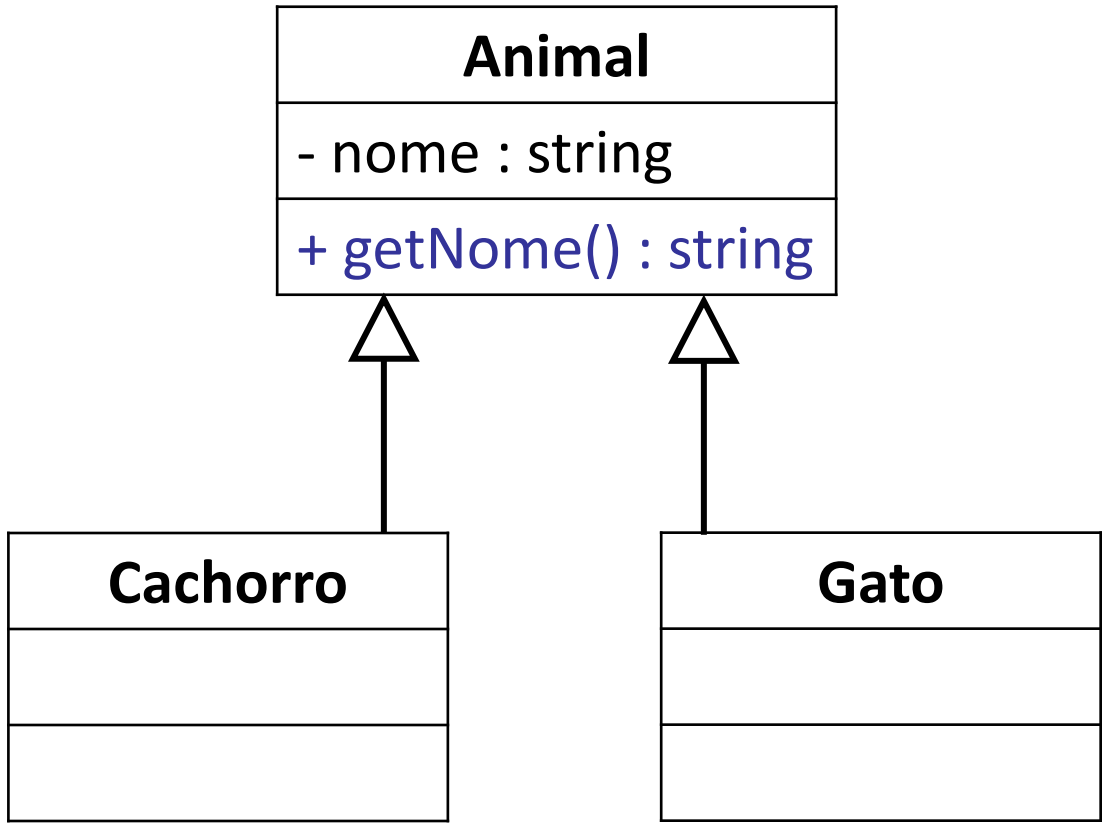
```
class Gato : Animal
{
   public :

};
```

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 :

};
```

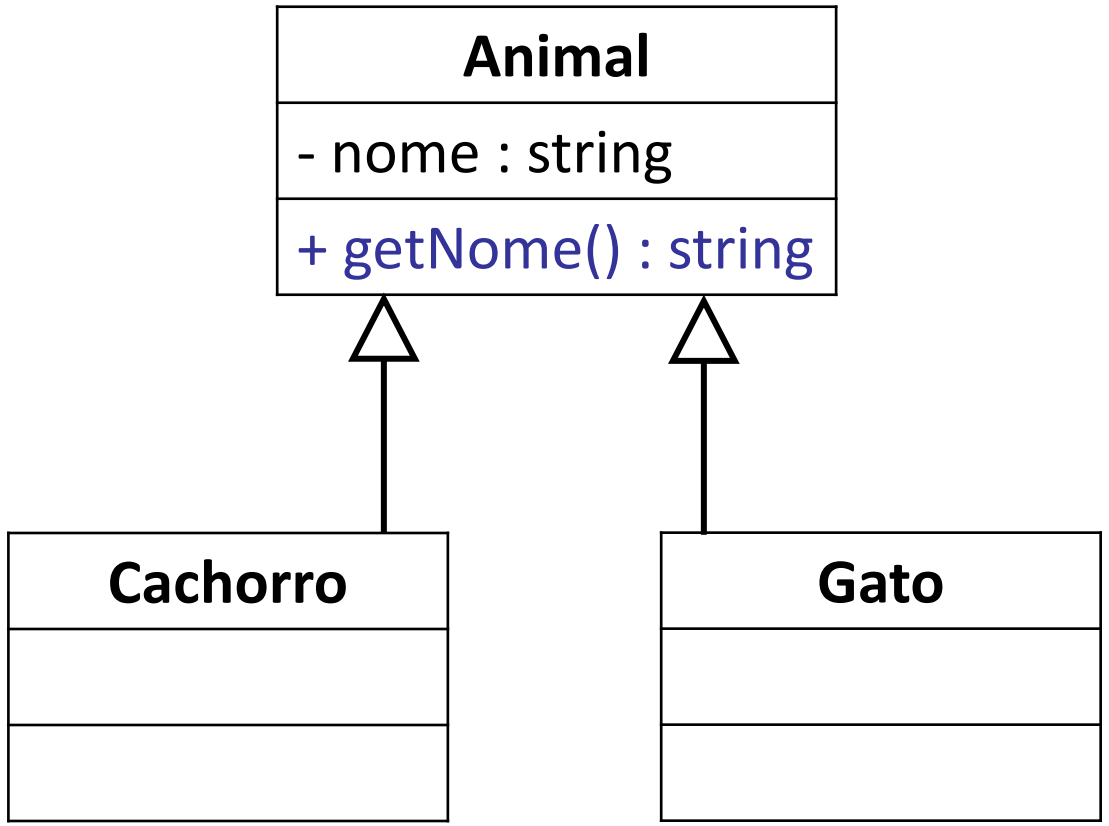
```
class Gato : Animal
{
   public :

};
```

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 :

};
```

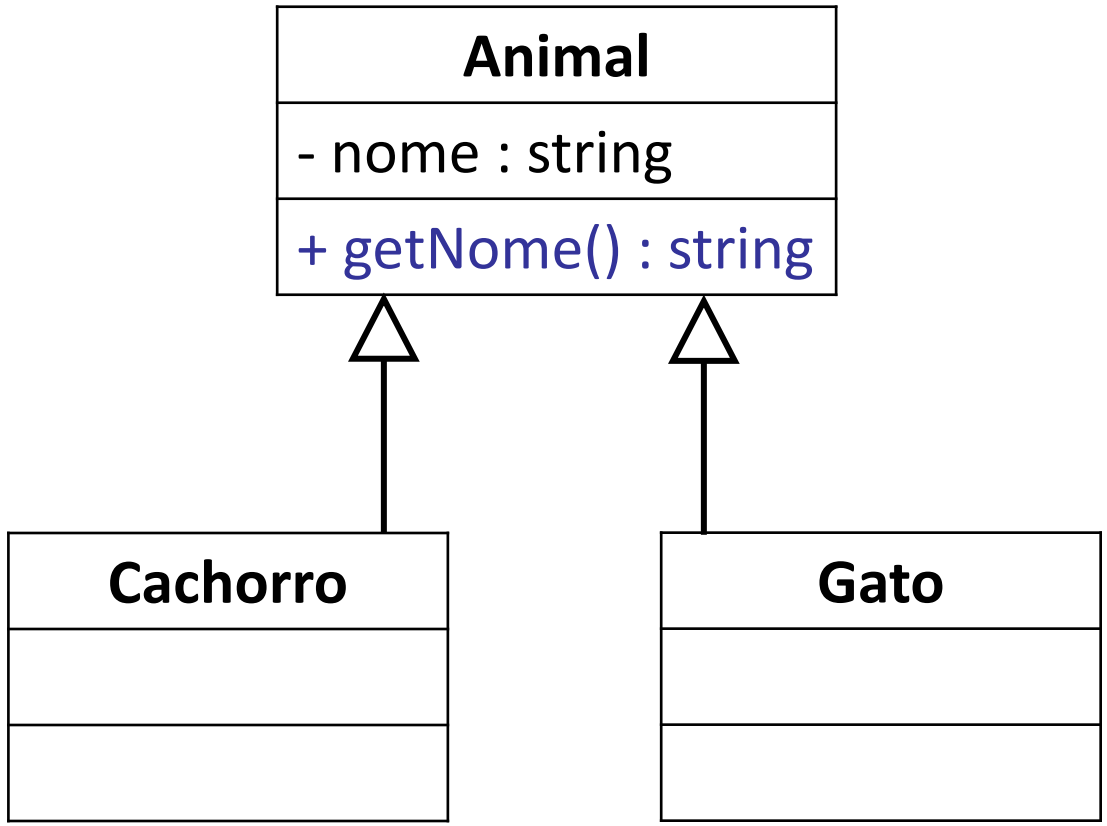
```
class Gato : Animal
{
   public :

};
```

animal

cachorro

gato

Generalização

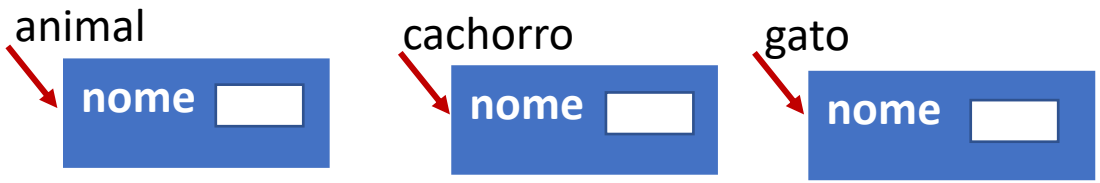



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

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

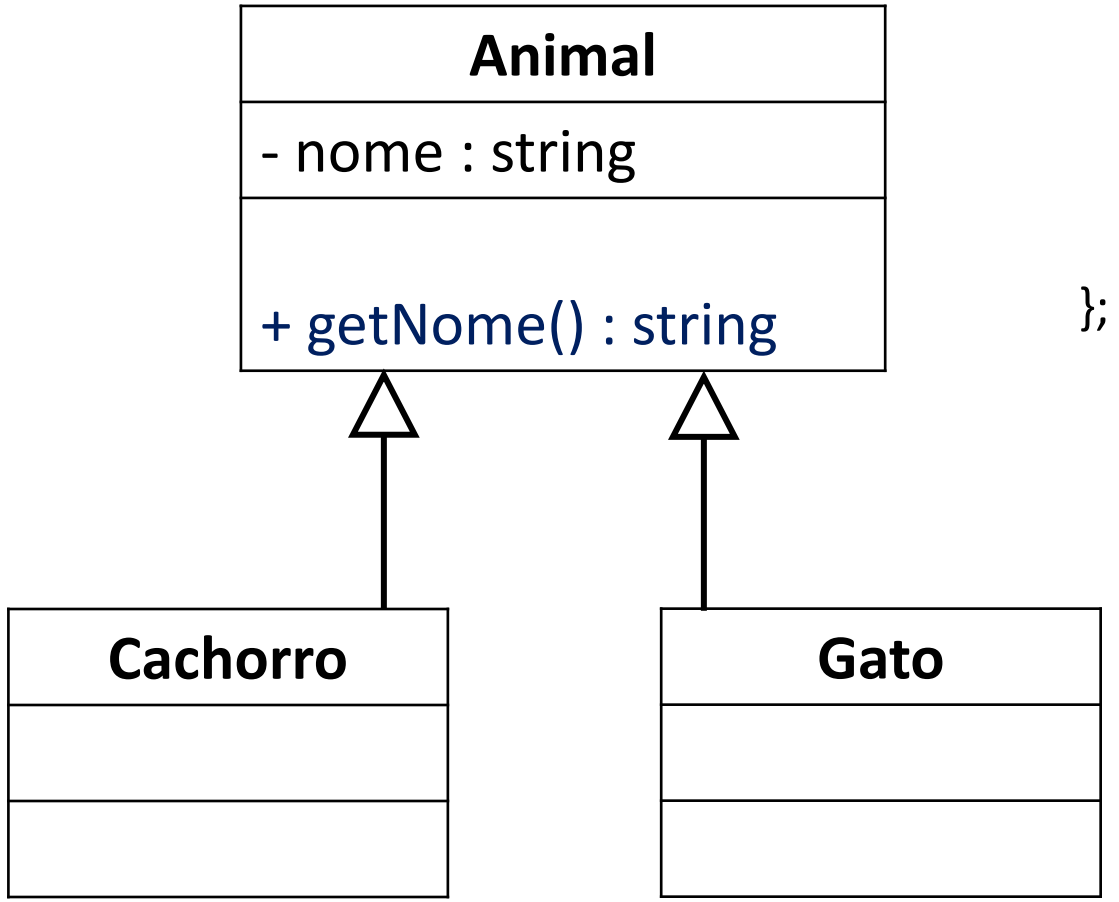
```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Construtores em casos de herança

Generalização

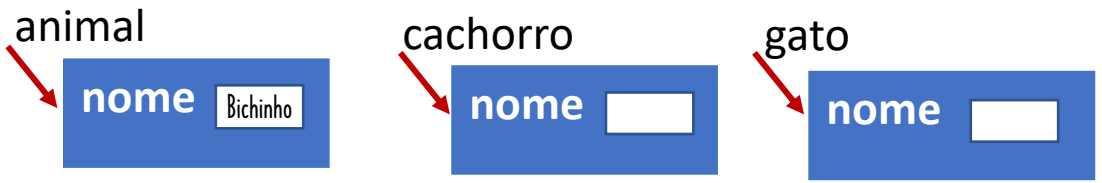



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

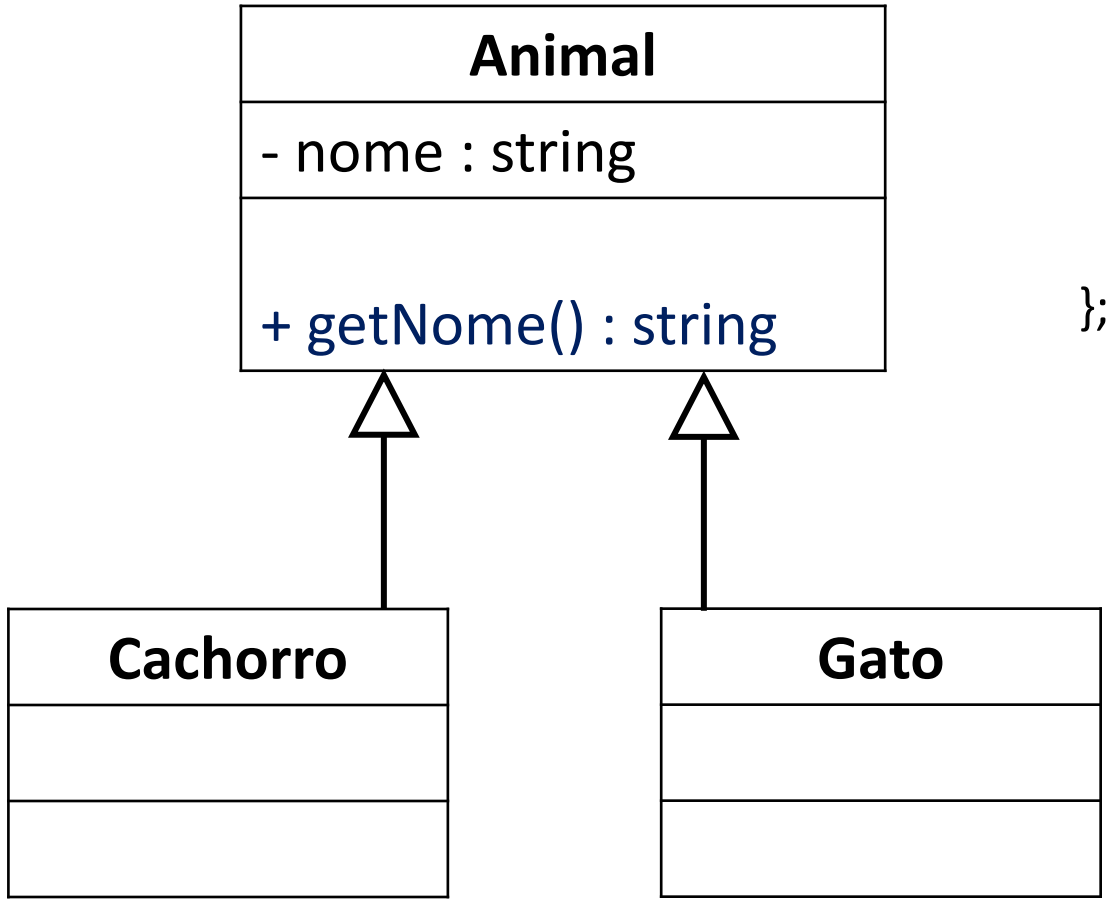
```
...
Animal* animal;
animal = new Animal("Bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

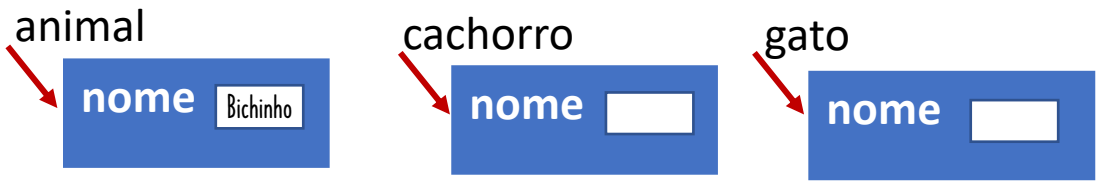



```
class Animal
{
   private :
        string nome;
   public :
        string getNome()
        {
            return nome;
        }
};
```

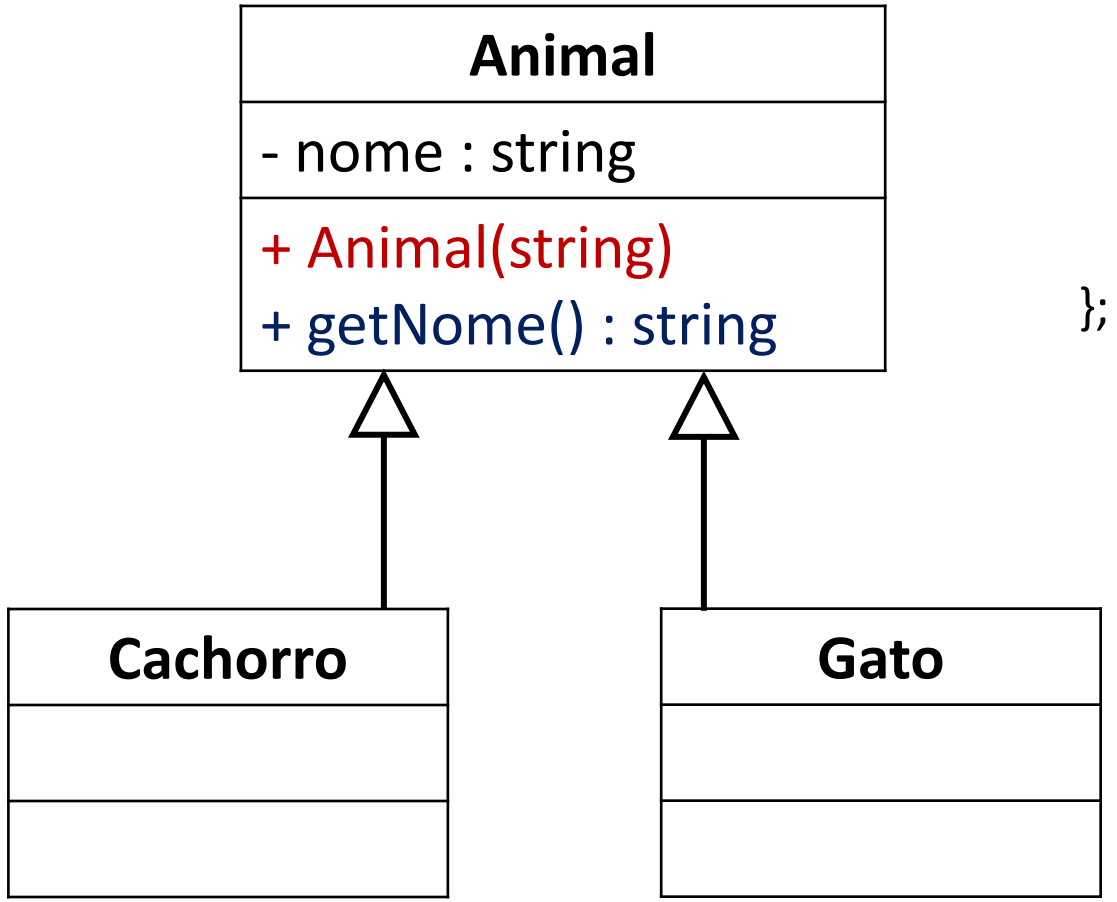
```
...
Animal* animal;
animal = new Animal("Bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

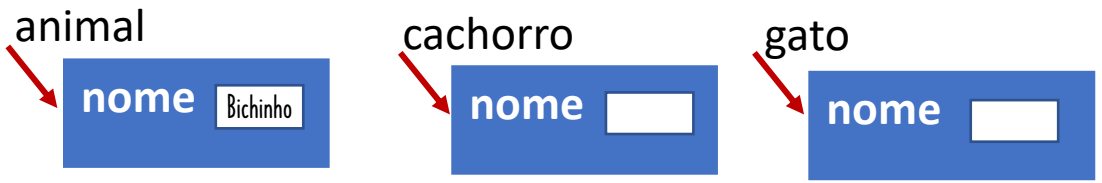



```
class Animal
{
   private :
      string nome;
   public :
      string getNome()
      {
         return nome;
      }
};
```

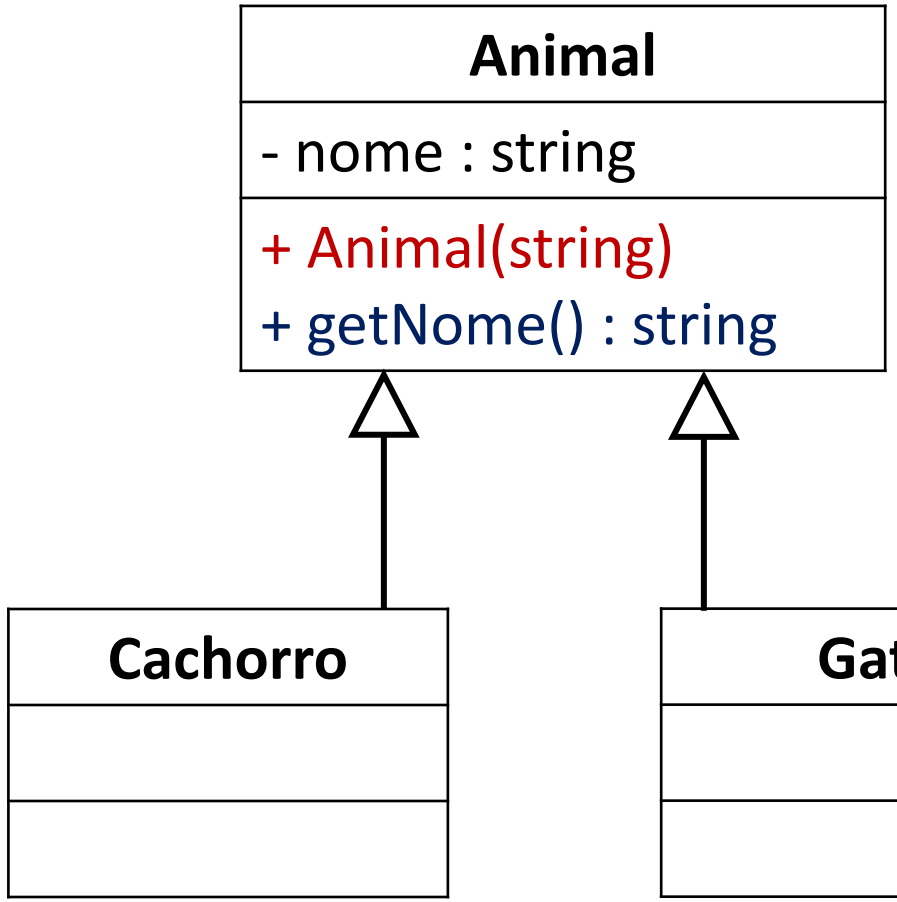
```
...
Animal* animal;
animal = new Animal("Bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro;
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

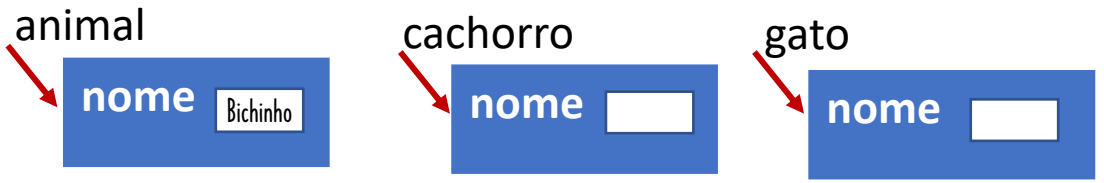



```
class Animal
{
   private :
        string nome;
   public :
        Animal(string nome)
        {
             this->nome= nome;
        }
        string getNome()
        {
            return nome;
        }
};
```

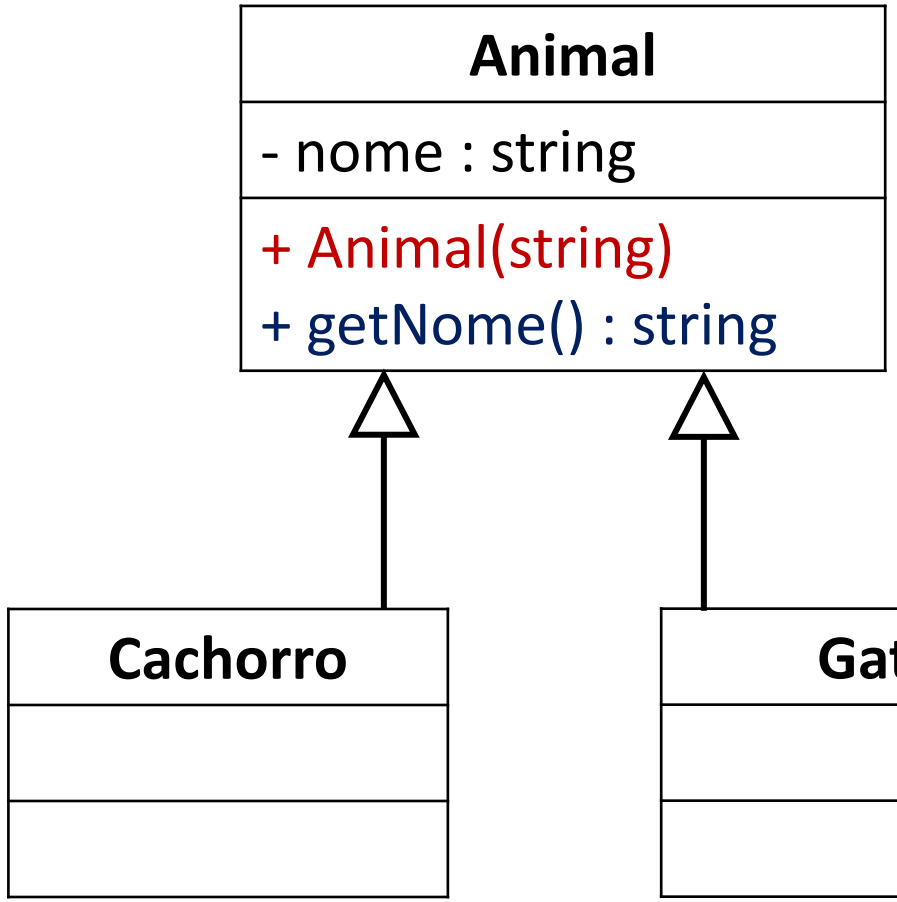
```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```

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



Generalização

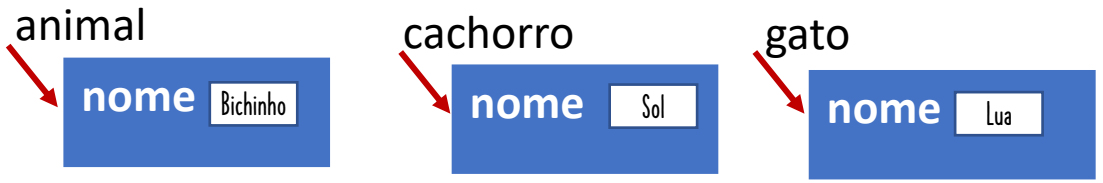



```
class Animal
{
   private :
      string nome;
   public :
      Animal(string nome)
      {
          this->nome= nome;
      }
      string getNome()
      {
          return nome;
      }
};
```

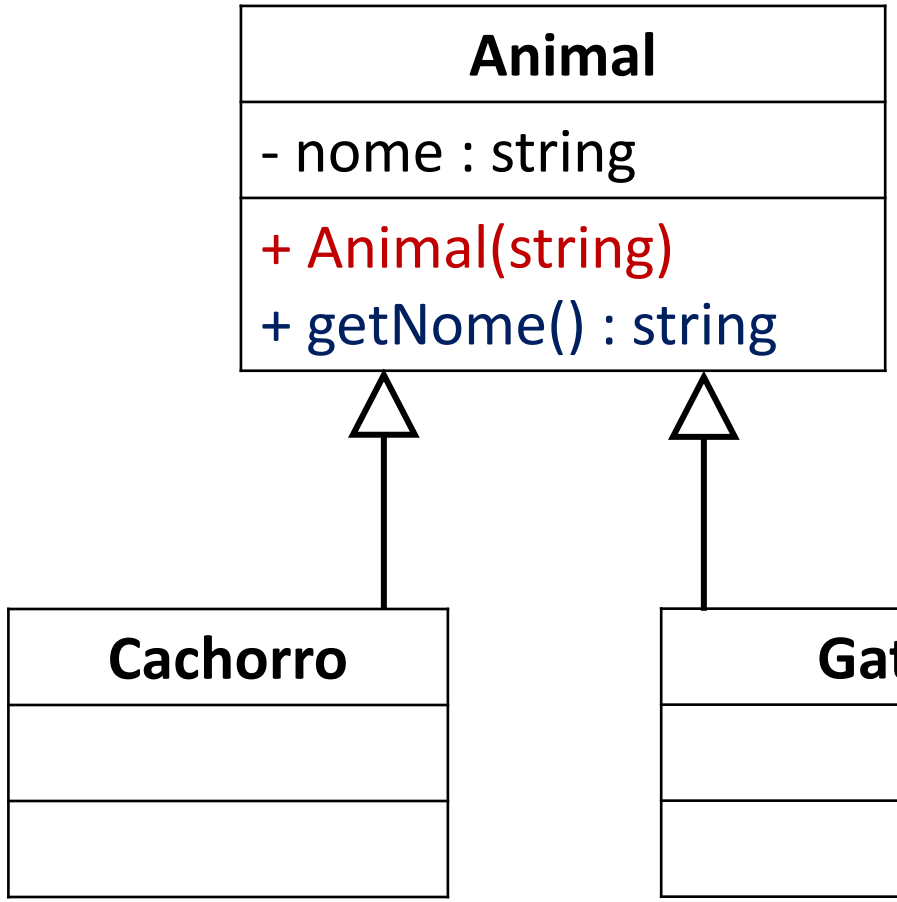
```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```

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



Generalização

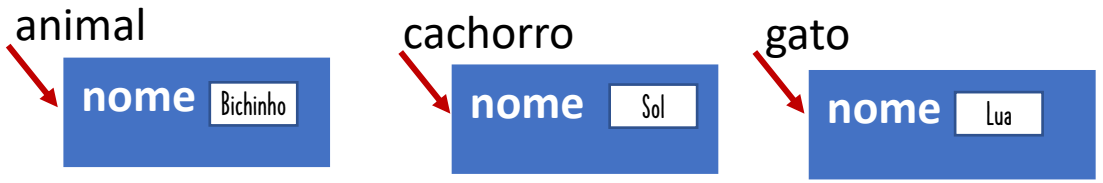



```
class Animal
{
  private :
    string nome;
  public :
    Animal(string nome)
    {
        this->nome= nome;
    }
    string getNome()
    {
       return nome;
    }
};
```

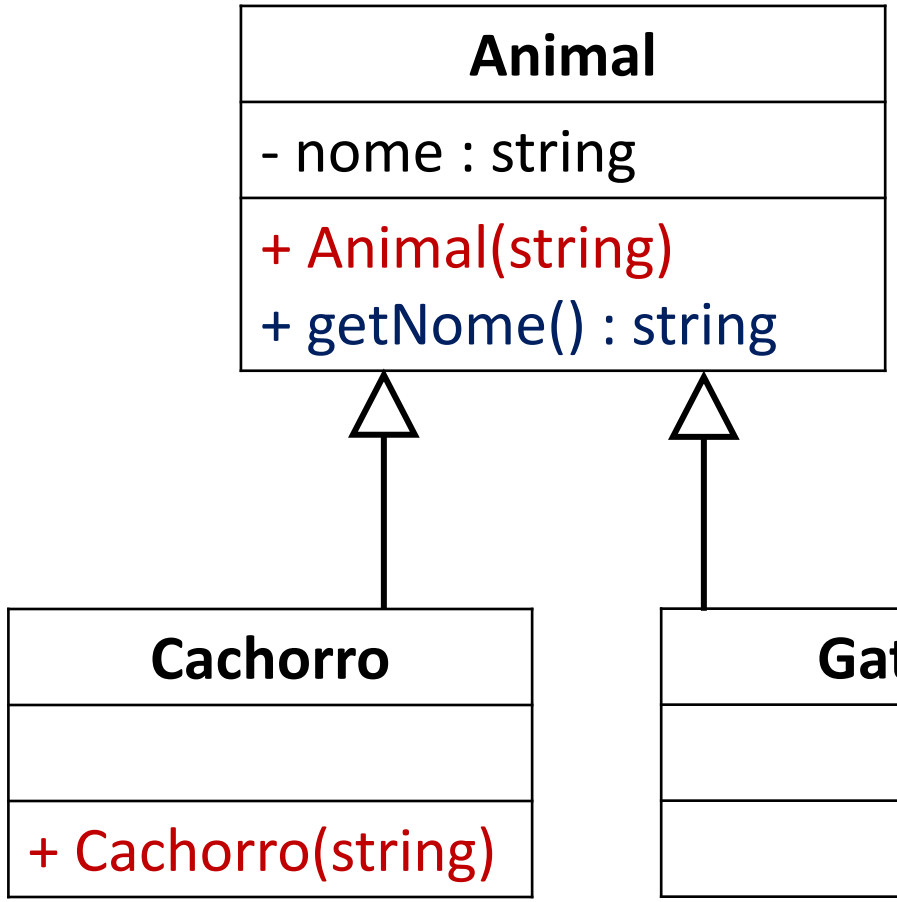
```
class Cachorro : Animal
{
   public :

};
```

```
class Gato : Animal
{
   public :

};
```

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



Generalização

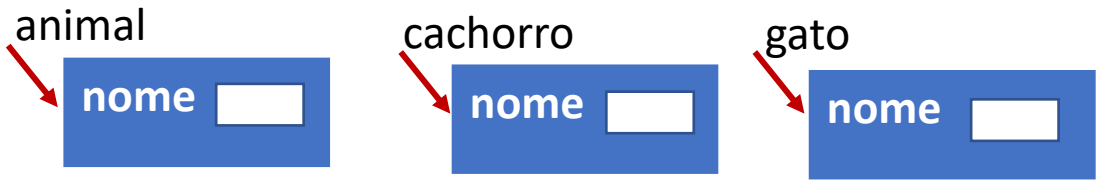



```
class Animal
{
  private :
    string nome;
  public :
    Animal(string nome)
    {
        this->nome= nome;
    }
    string getNome()
    {
        return nome;
    }
};
```

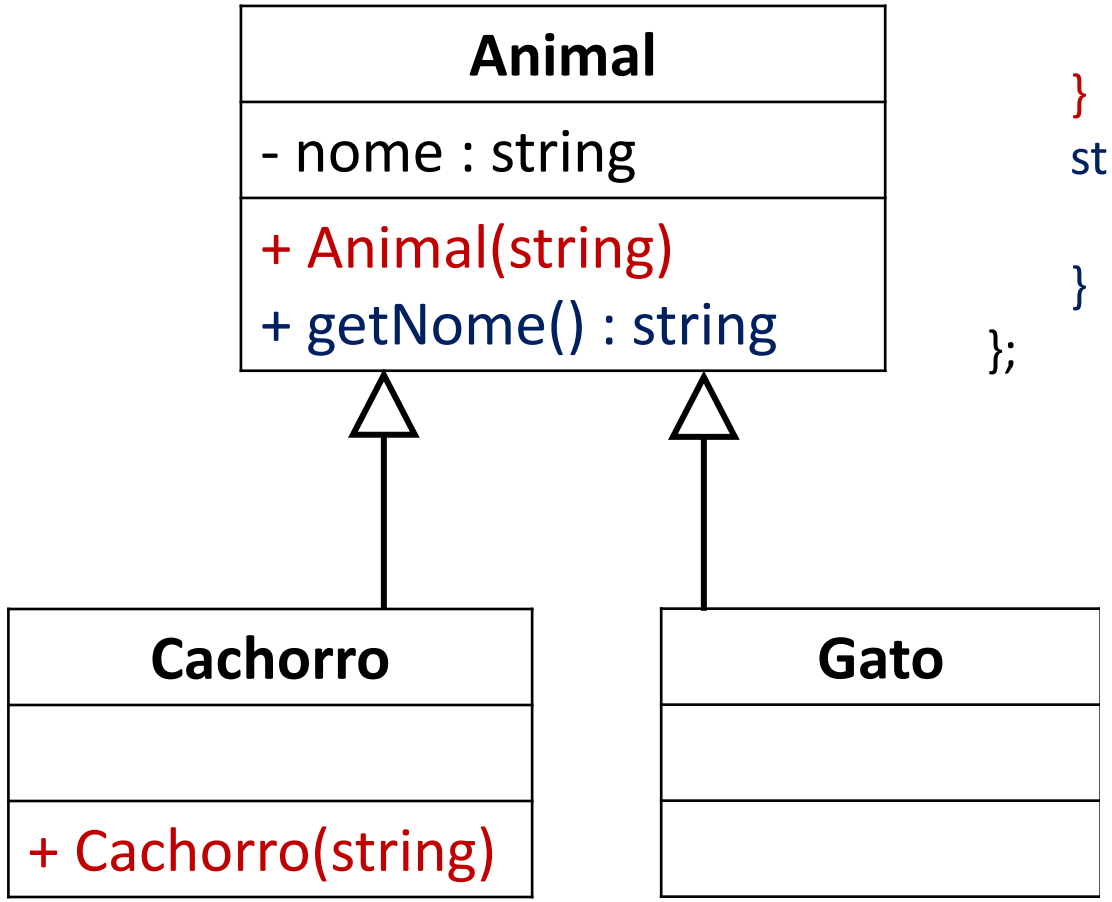
```
class Cachorro : Animal
{
  public :

};
```

```
class Gato : Animal
{
   public :

};
```

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



Generalização

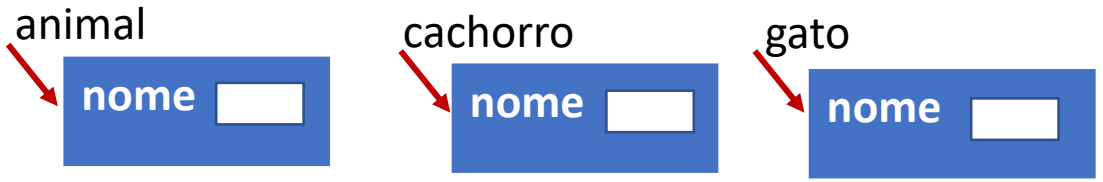



```
class Animal {
  private :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

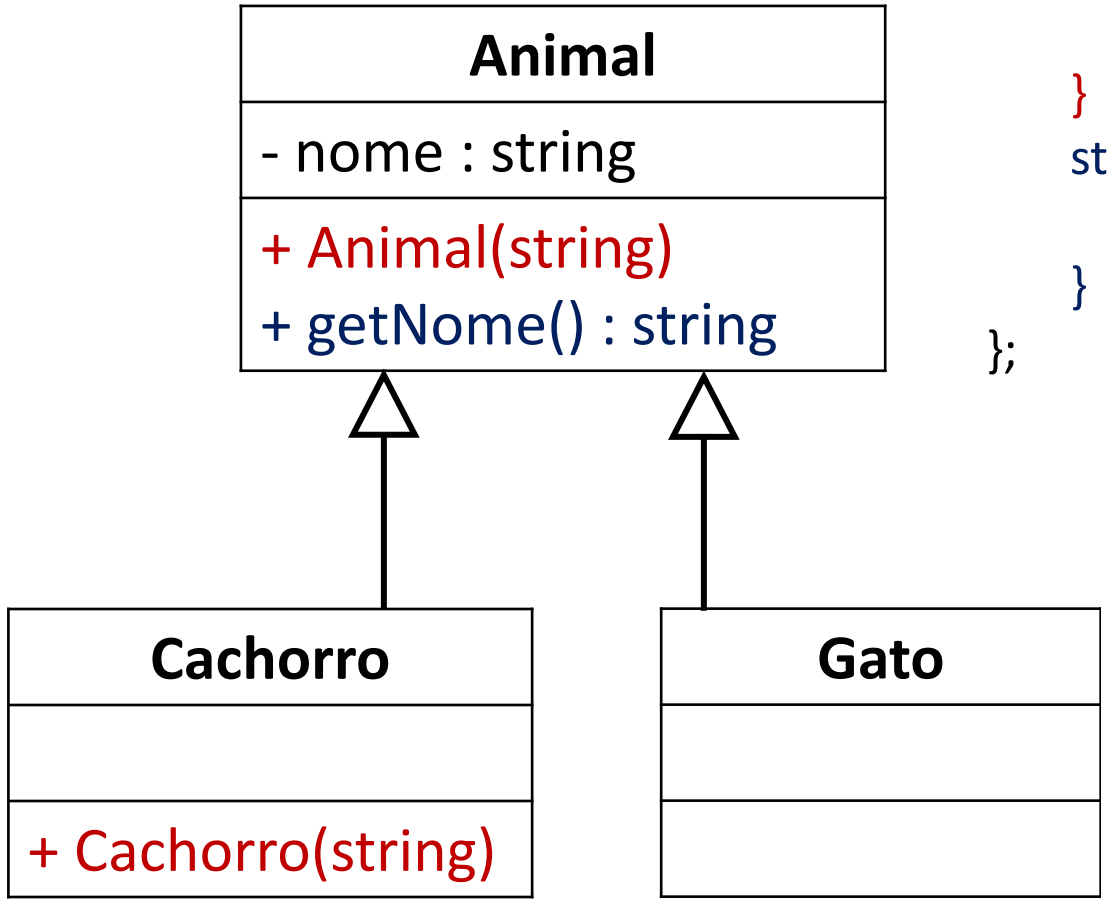
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

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

```
class Gato : Animal
{
   public :

};
```



Generalização

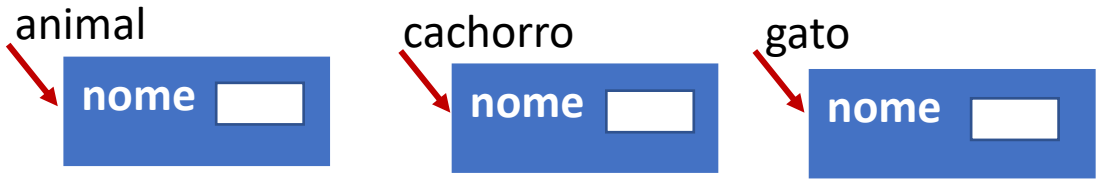



```
class Animal {
  private :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

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

```
class Cachorro : Animal
{
  public :
    Cachorro(string nome)
    {
      this->nome= nome; //erro
    }
};
```

```
class Gato : Animal
{
   public :

};
```



Escopo de visibilidade

Público:

Visível a todos que têm acesso ao objeto

Privado:

Visível apenas aos elementos da mesma classe

Protegido:

Visível apenas aos elementos da mesma classe e aos de suas especializações

Uma alternativa para o problema

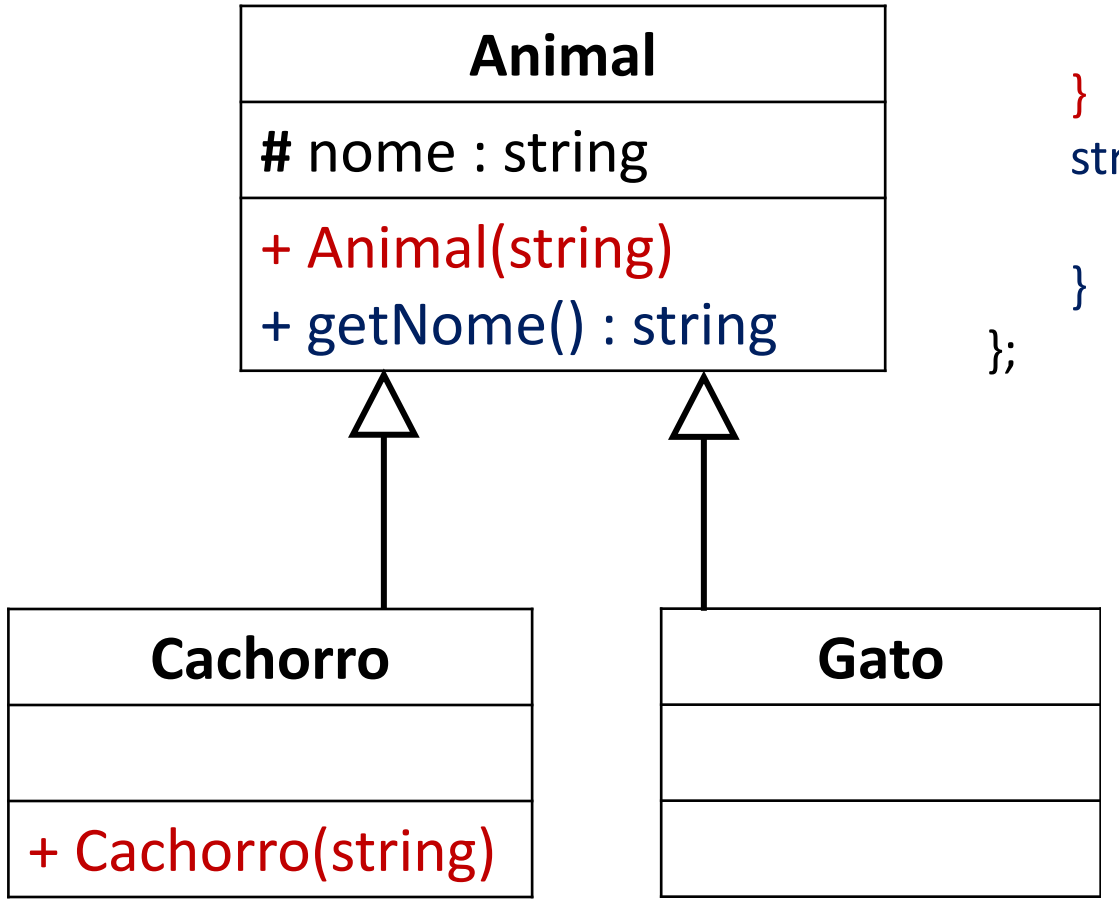
Assim,

uma alternativa para o problema é o atributo **nome**, da classe-base, ter o escopo de sua visibilidade definido como **protected**

Animal
nome : string
+ Animal(string nome); + getNome() : string

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

Generalização

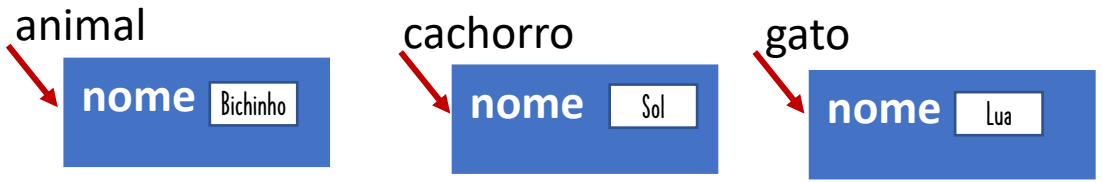



```
class Animal {
  protected :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

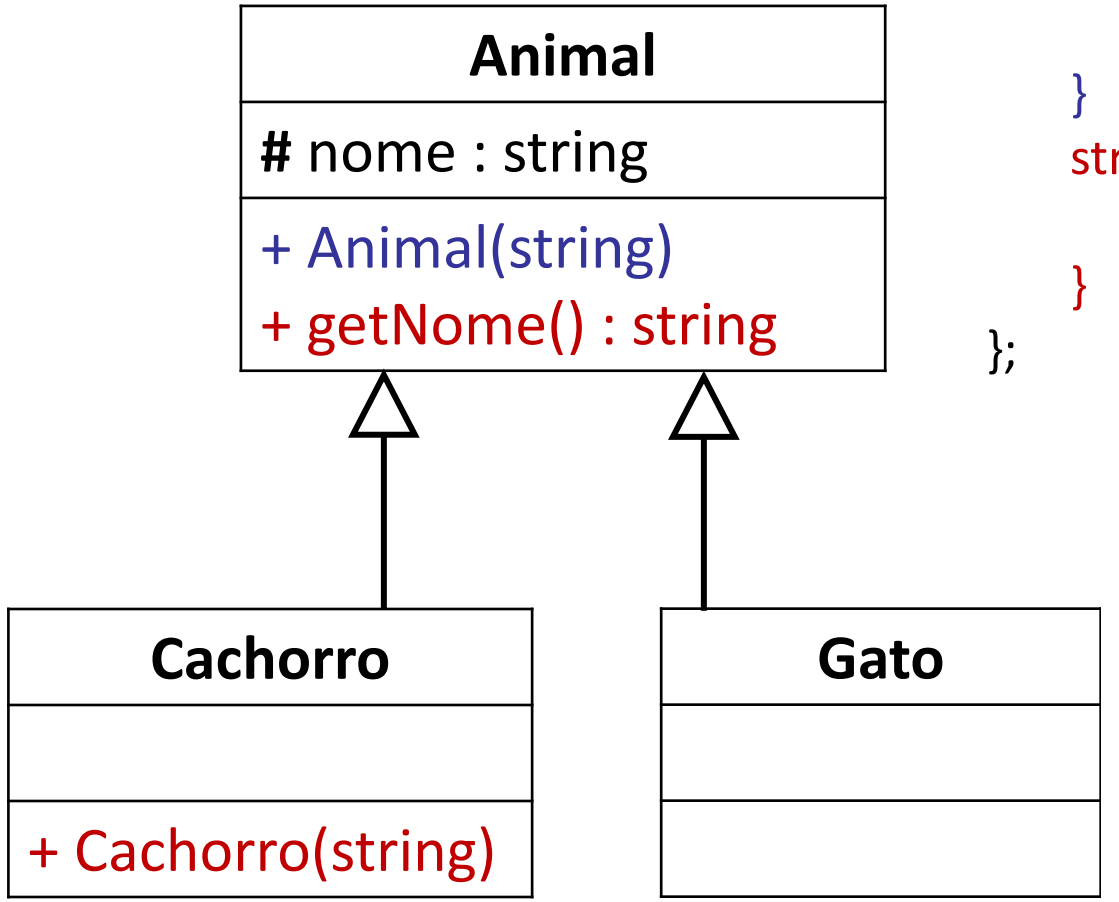
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
```

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

```
class Gato : Animal
{
   public :

};
```



Generalização

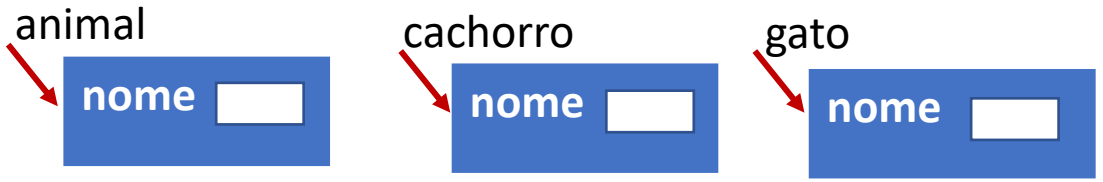



```
class Animal {
  protected :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

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

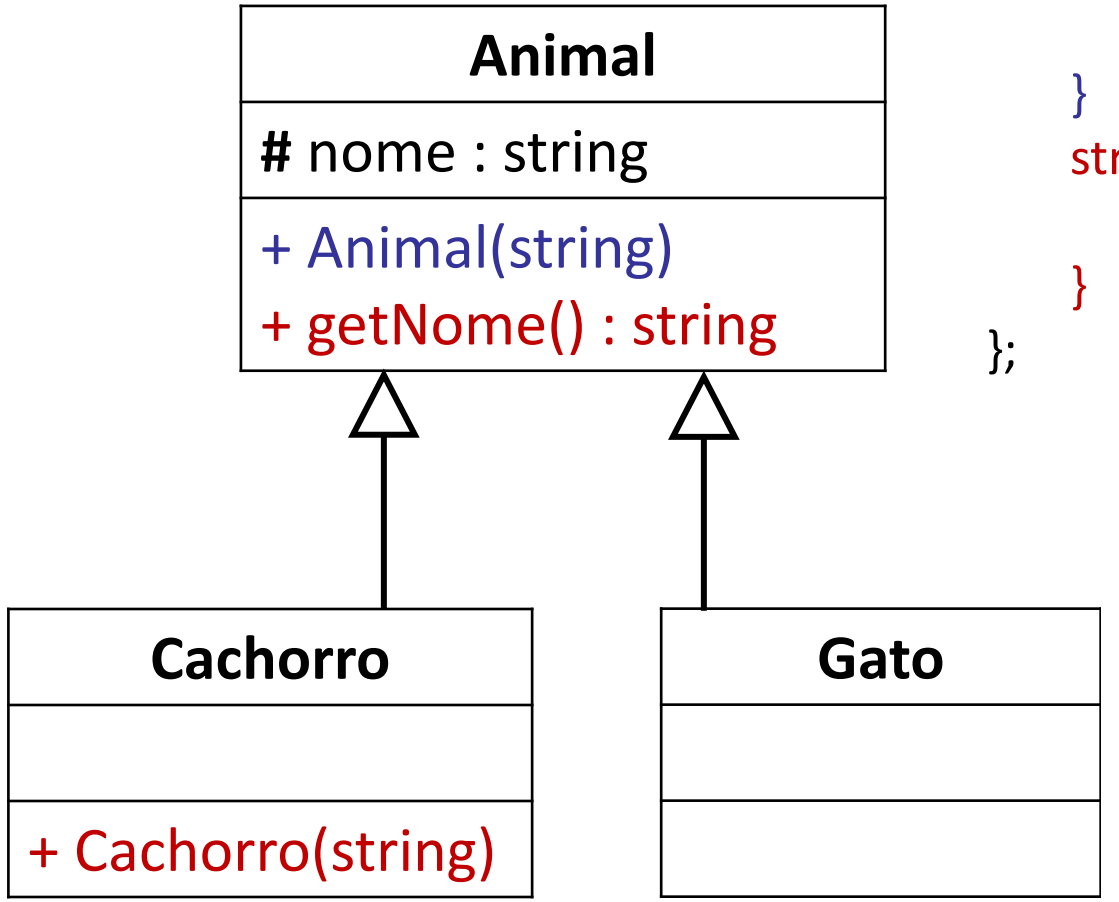
```
class Gato : Animal
{
   public :

};
```

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



Será autorizada a execução do método público *getNome()* sobre a instância da classe Cachorro?

Generalização

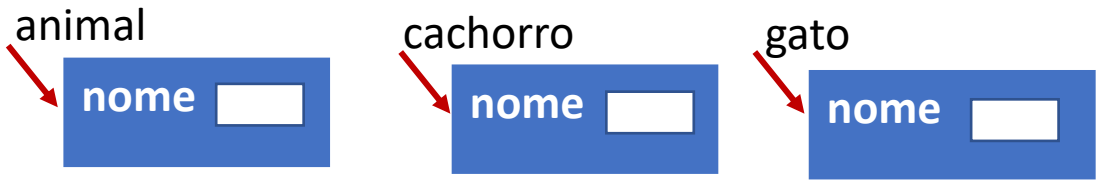



```
class Animal {
  protected :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

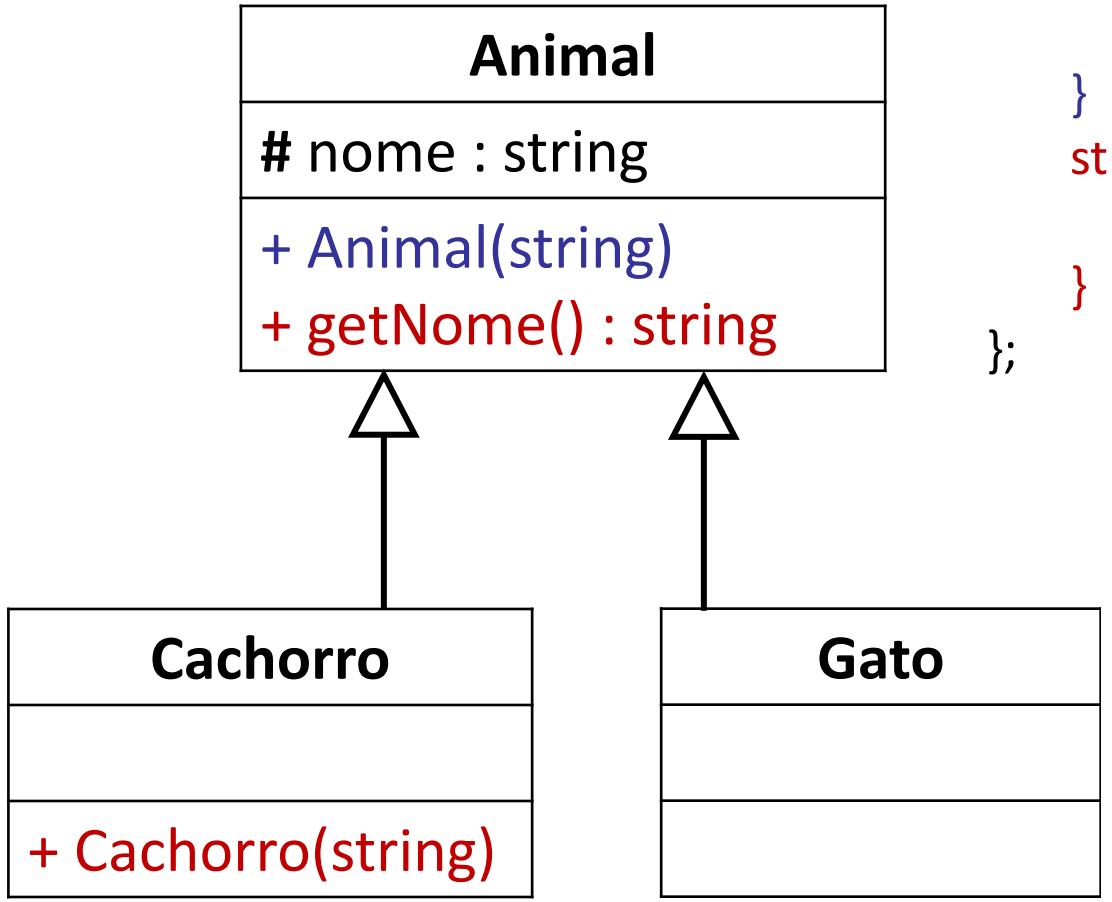
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
cout << cachorro->getNome(); //erro
```

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

```
class Gato : Animal
{
   public :

};
```



Generalização

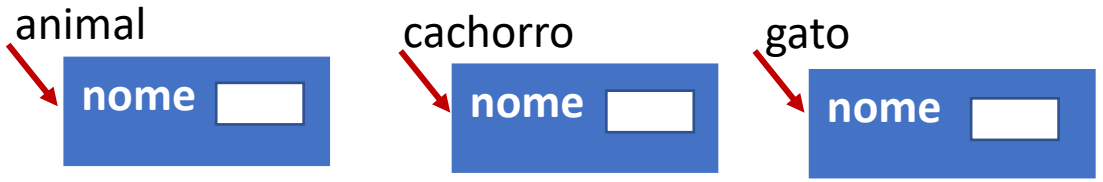



```
class Animal {
  protected :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

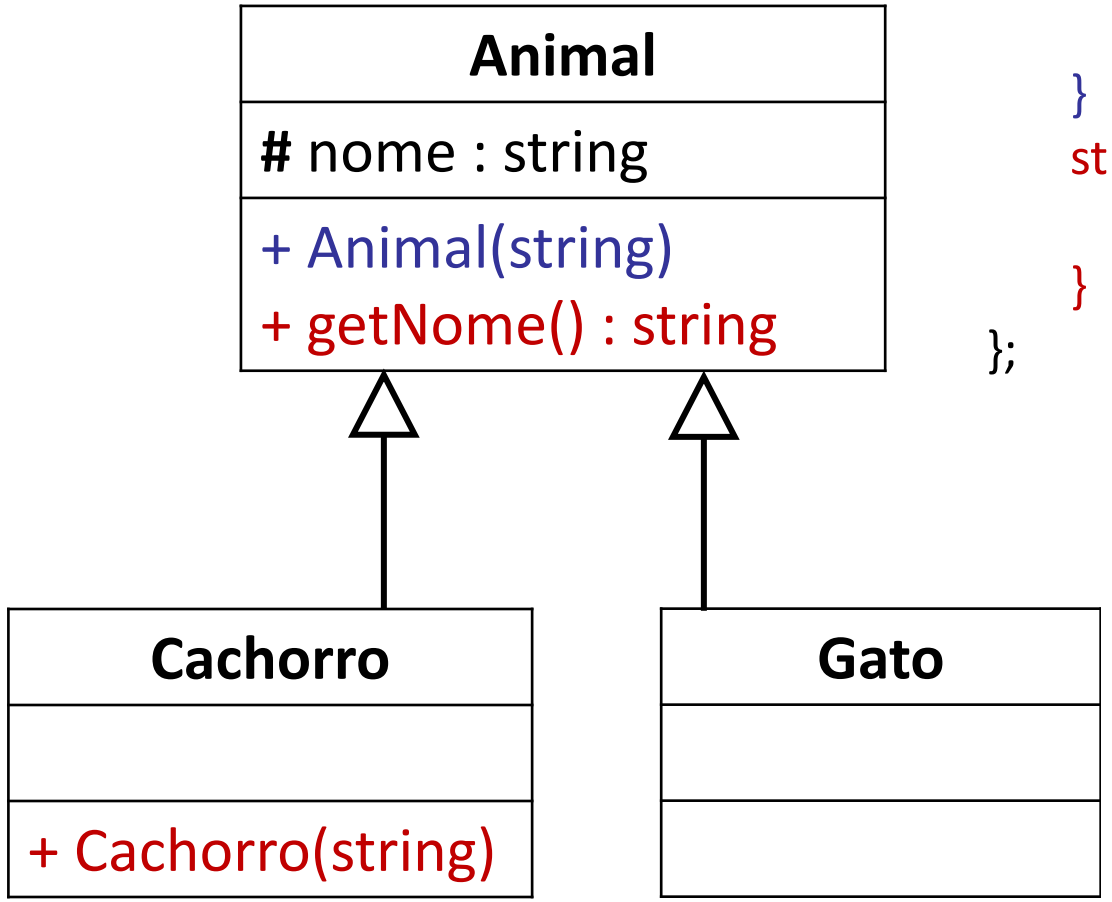
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
cout << cachorro->getNome(); //Herança
```

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

```
class Gato : Animal
{
   public :

};
```



Generalização

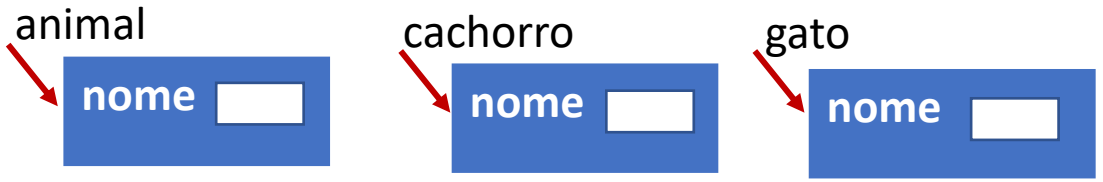



```
class Animal {
  protected :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

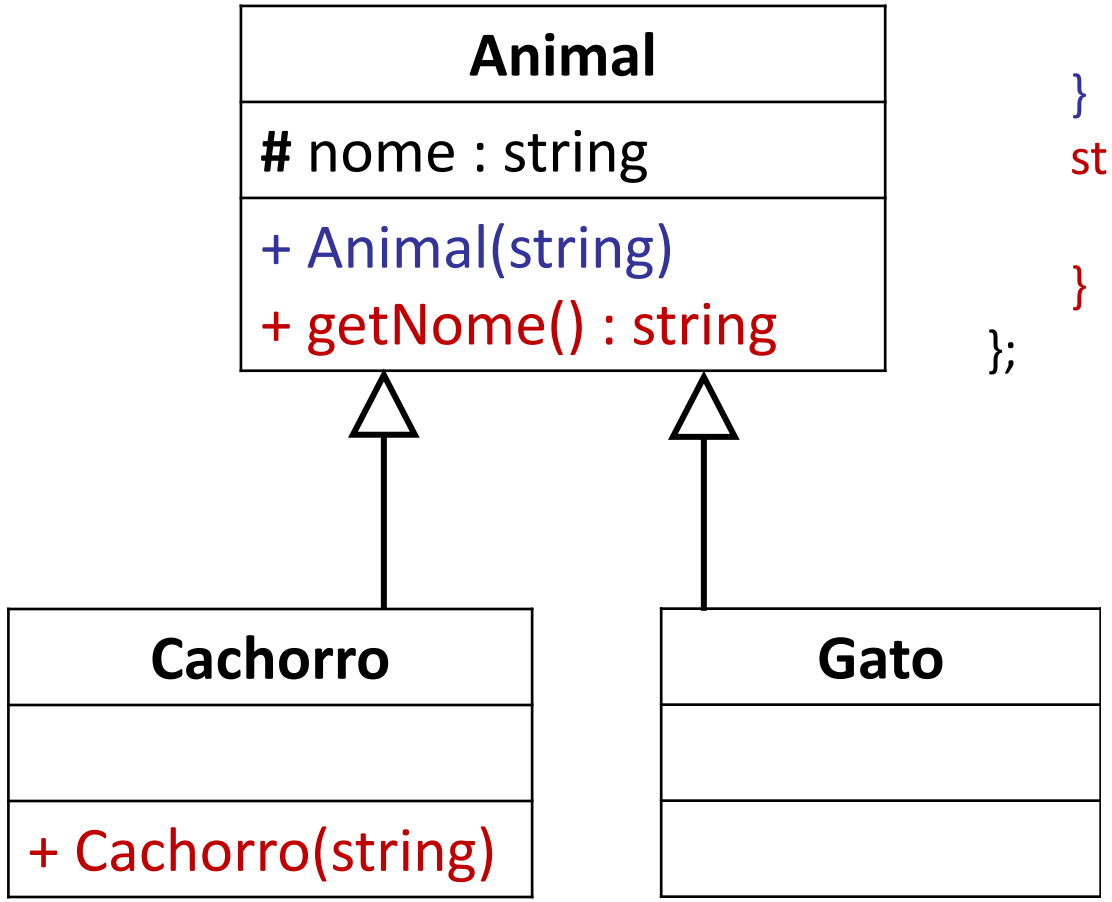
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
cout << cachorro->getNome(); //erro
```

```
class Cachorro : Animal // herança privada
{
  public :
     Cachorro(string nome)
     {
       this->nome= nome;
     }
};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

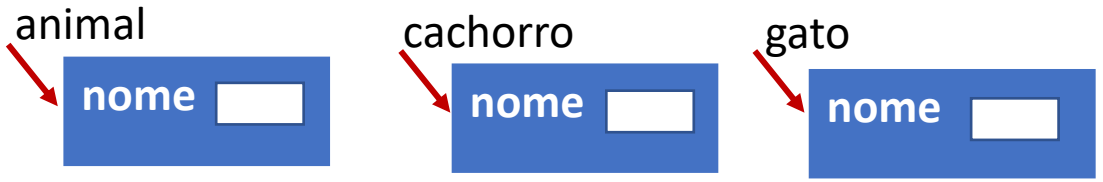



```
class Animal {
  protected :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

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

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

```
class Gato : public Animal
{
   public :

};
```

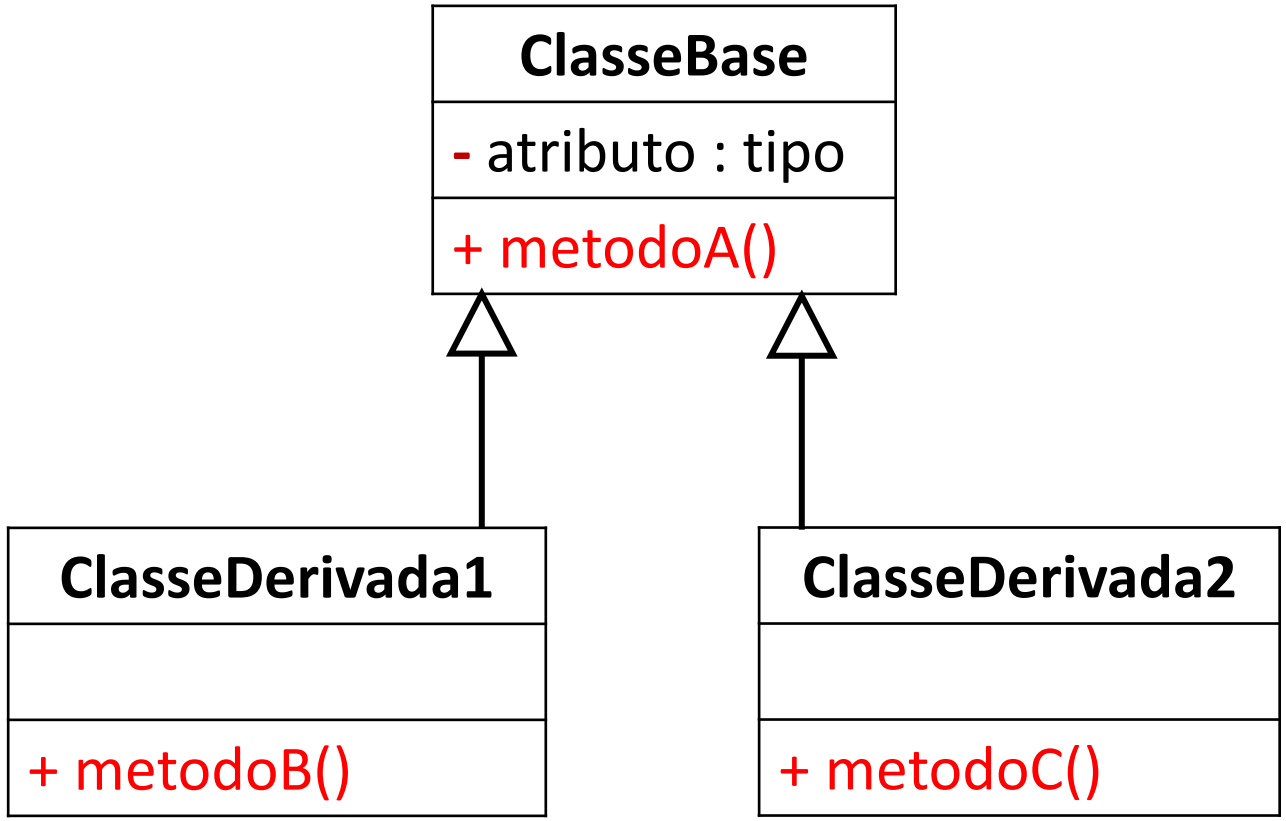


Escopo de visibilidade

E qual alguma outra alternativa para o caso de manter o atributo com escopo de visibilidade igual a privado?

Escopo de visibilidade

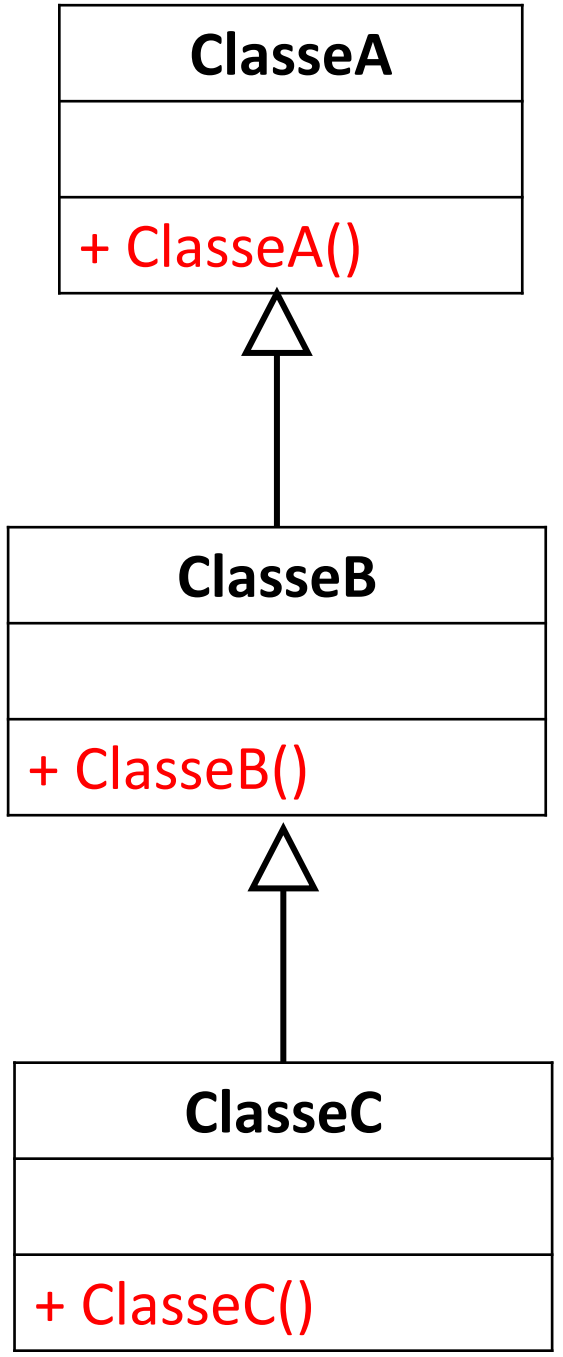
Para o caso de manter o elemento privado, há alguma outra alternativa para acessá-lo?



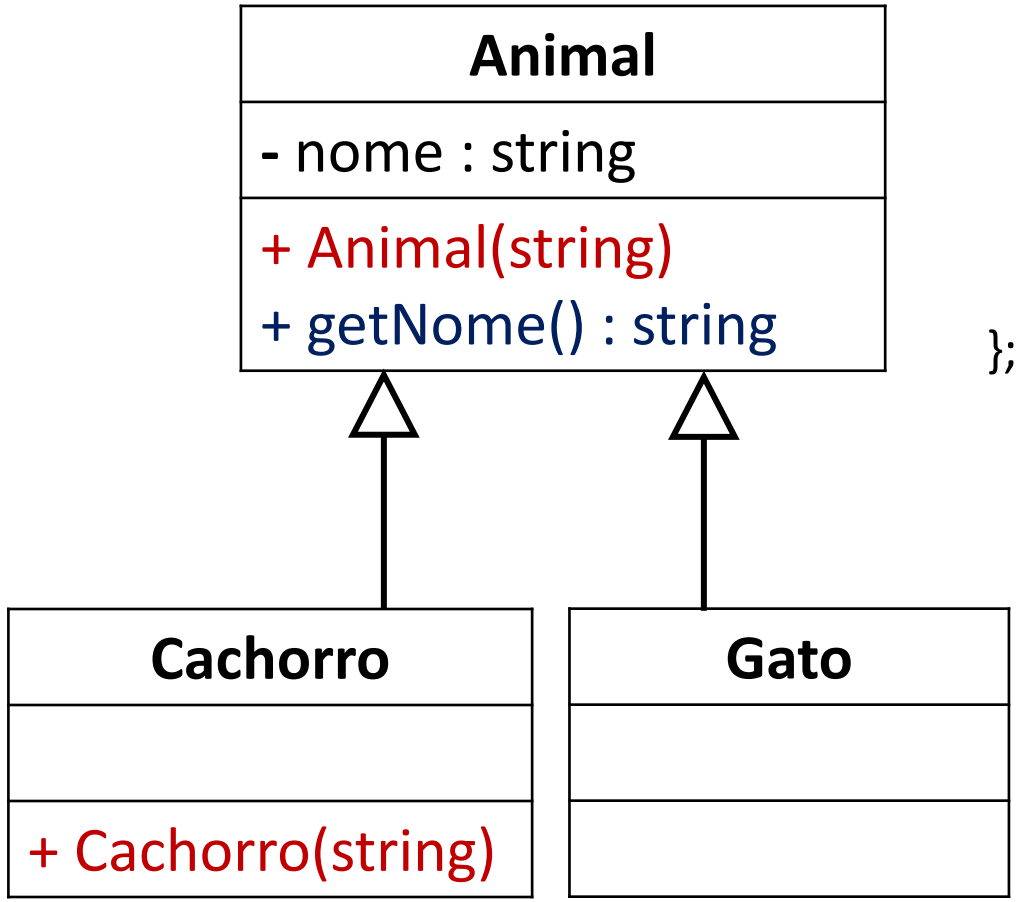
Construtores em hierarquia

E qual alguma outra alternativa para o caso de manter o atributo com escopo de visibilidade igual a privado?

Um construtor pode, explicitamente, solicitar a execução do construtor de sua classe base (superclasse).



Generalização

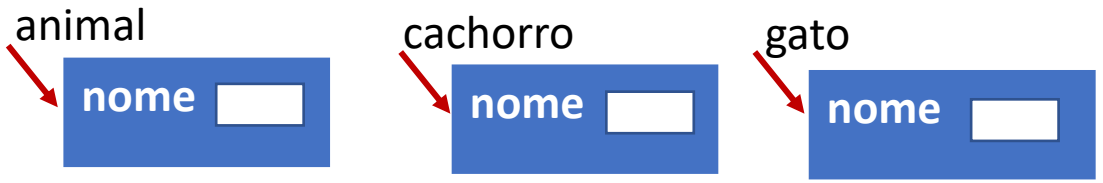



```
class Animal {
  private :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

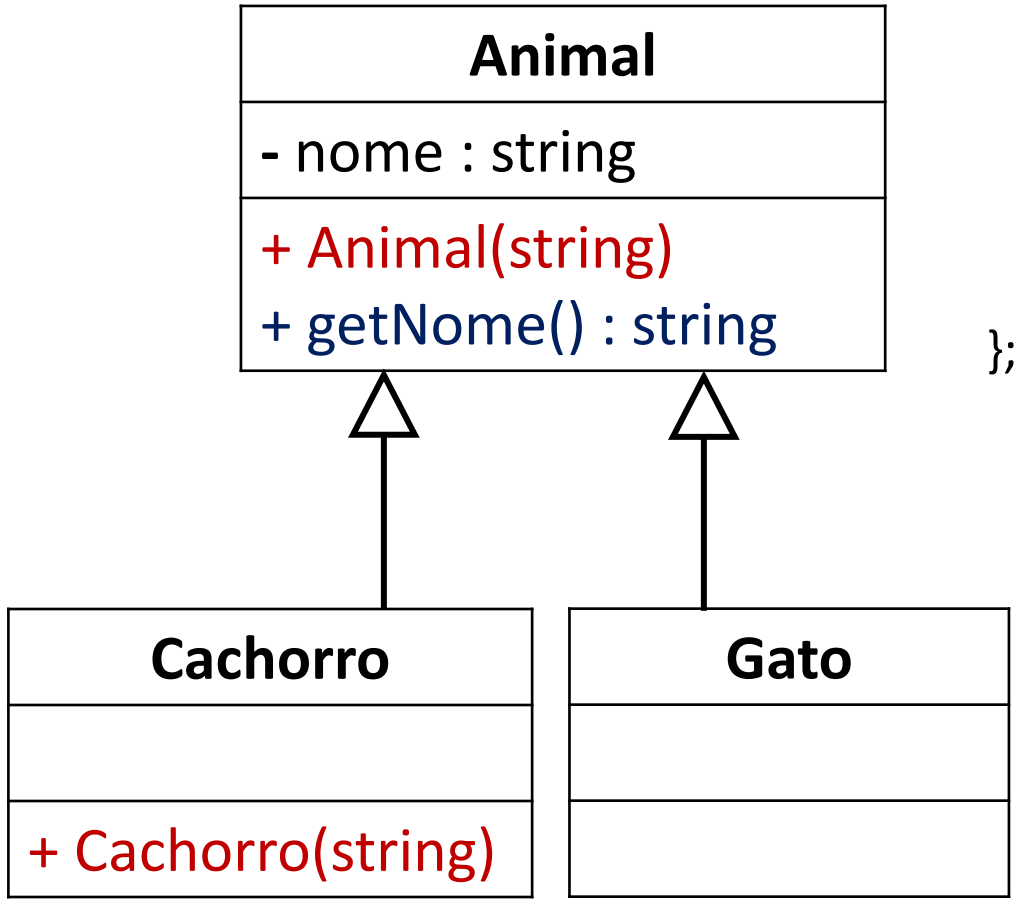
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato;
...
cout << animal->getNome();
cout << cachorro->getNome();
```

```
class Cachorro : public Animal
{
  public :
    Cachorro(string nome) : Animal(nome)
    {  }
};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

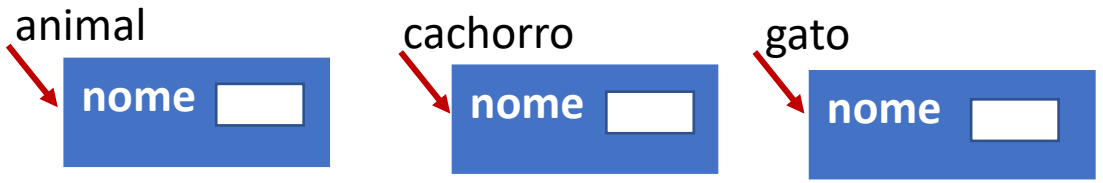



```
class Animal {
  private :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

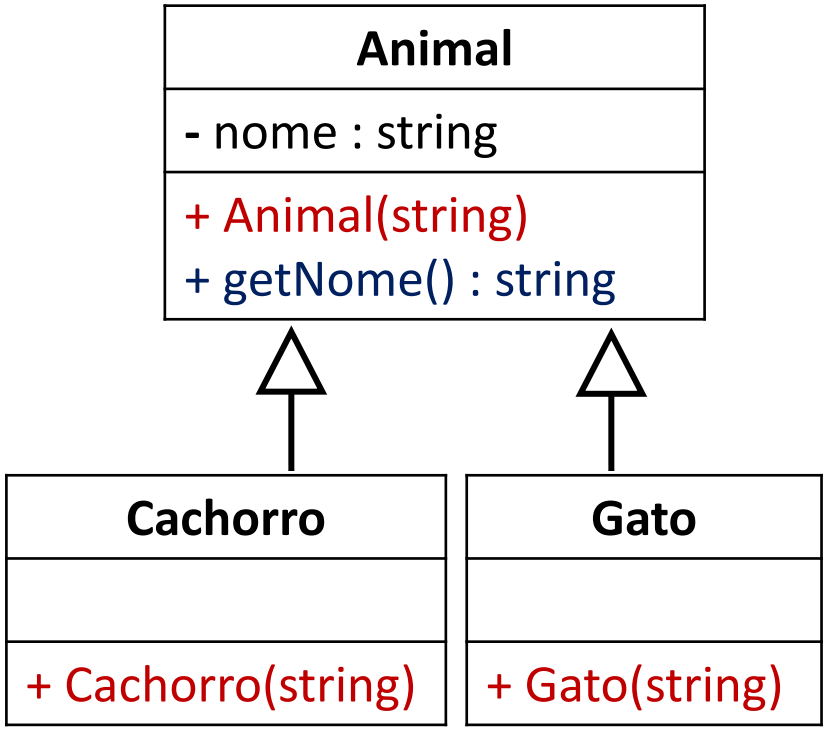
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato("lua");
...
cout << animal->getNome();
cout << cachorro->getNome();
cout << gato->getNome();
```

```
class Cachorro : public Animal
{
  public :
     Cachorro(string nome) : Animal(nome)
     {  }
};
```

```
class Gato : Animal
{
   public :

};
```



Generalização

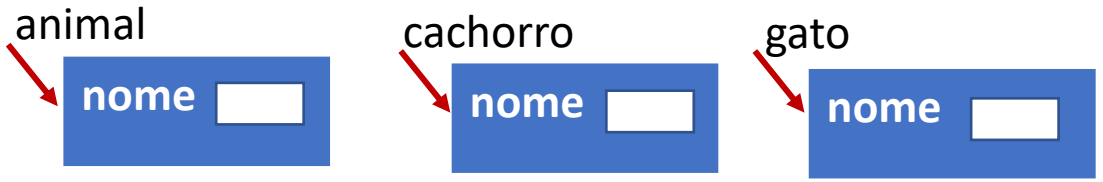



```
class Animal {
  private :
     string nome;
  public :
     Animal(string nome) {
          this->nome= nome;
     }
     string getNome() {
          return nome;
     }
};
```

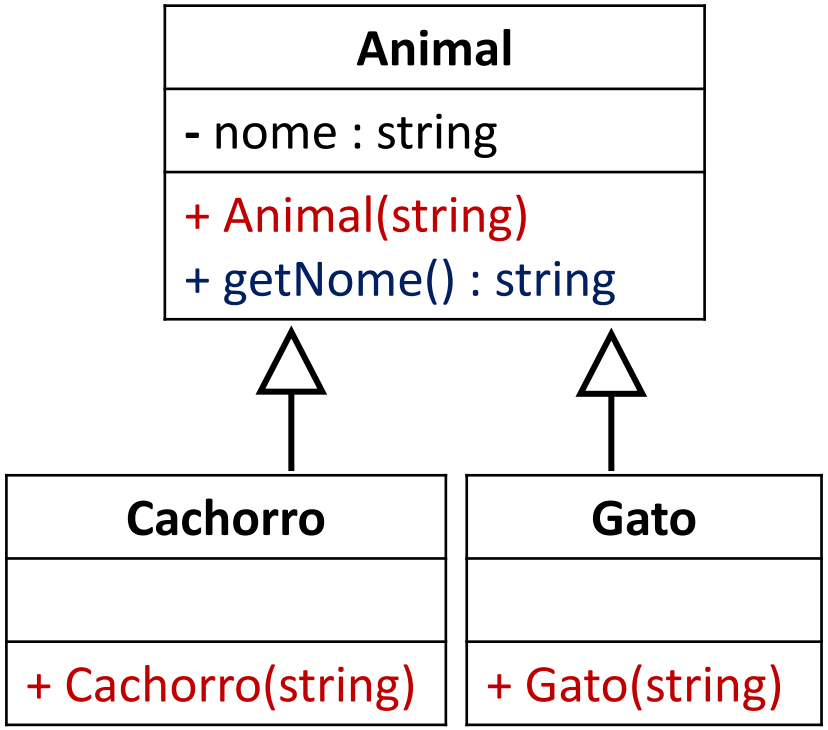
```
...
Animal* animal;
animal = new Animal("bichinho");
...
Cachorro * cachorro;
cachorro = new Cachorro("sol");
...
Gato * gato;
gato = new Gato("lua");
...
cout << animal->getNome();
cout << cachorro->getNome();
cout << gato->getNome();
```

```
class Cachorro : public Animal {
  public :
     Cachorro(string nome) : Animal(nome)
     {  }
};
```

```
class Gato : public Animal
{
   public :

};
```



Generalização

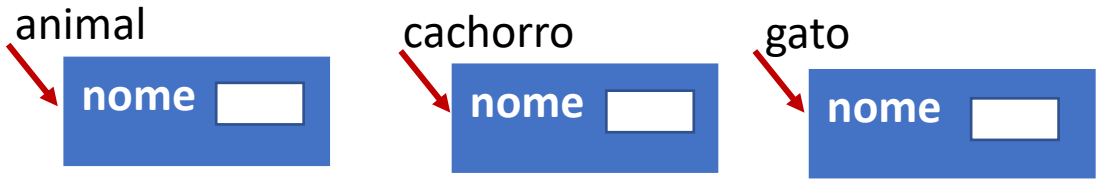



```
class Cachorro : public Animal {
  public :
    Cachorro(string nome) : Animal(nome)
    {  }
};
```

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

```
class Gato : public Animal {
   public :
      Gato(string nome) : Animal(nome)
      {  }
};
```

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



Trabalho Prático Final
Implementar Generalização

Menu de Opções

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

7 – Aniversariantes do mês

Submenu a cada opção do menu principal

1 - Cadastrar uma pessoa
    1.1 - Cadastrar Professor
    1.2 - Cadastrar Aluno
2 - Listar todas as pessoas
    2.1 – Listar Professores
    2.2 – Listar Alunos
3 – Pesquisar por nome
    3.1 – Pesquisar Professores por nome
    3.2 – Pesquisar Alunos por nome
4 – Pesquisar por CPF
    4.1 – Pesquisar Professores por CPF
    4.2 – Pesquisar Alunos por CPF
5 – Excluir pessoa
    5.1 – Excluir Professor (pelo CPF)
    5.2 – Excluir Aluno (pelo CPF)
6 - Apagar todas as pessoas cadastradas
    6.1 – Excluir todos os Professores
    6.2 – Excluir todos os Alunos
7 – Aniversariantes do mês
    7.1 – Informar o mês a ser pesquisado
    7.1 – Listar os Professores aniversariantes do mês
    7.2 – Listar os Alunos aniversariantes do mês