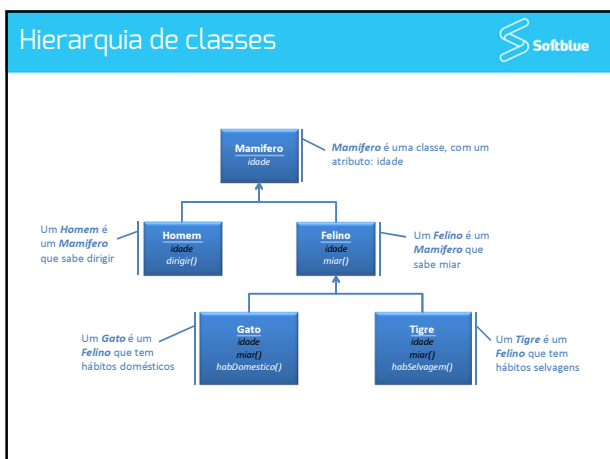




Herança

- Permite que uma classe filha (subclasse) herde atributos e métodos não privados (**private**) de sua classe pai, também conhecida como superclasse
- Benefícios:
 - Reaproveitamento de código
 - Organização do código
 - Facilidade de manutenção
- Em algumas linguagens é possível utilizar herança múltipla



Herdando uma classe

• Operador de herança (:)

```

class Mamifero
{
public:
    int idade;
    int peso;

    void Mamifero::andar() {
        cout << "andar";
    }
}

class Homem
{
public:
    int idade;
    int peso;
    int cpf;

    void Homem::andar() {
        cout << "andar";
    }

    void Homem::dirigir() {
        cout << "dirigir";
    }
}

class Homem : public Mamifero
{
public:
    int idade;
    int peso;
    int cpf;

    void Mamifero::andar() {
        cout << "andar";
    }

    void Homem::dirigir() {
        cout << "dirigir";
    }
}
  
```

Construtores hierárquicos

• Inicializando classes hierárquicamente

```

class Mamifero
{
public:
    int idade;
    int peso;

    Mamifero::Mamifero(int idade, int peso) {
        this->idade = idade;
        this->peso = peso;
    }

    void Mamifero::andar() {
        cout << "andar";
    }
}

class Homem : public Mamifero
{
public:
    int cpf;


    Homem::Homem(int idade, int peso, int cpf) : Mamifero(idade, peso) {
        this->cpf = cpf;
    }

    void Homem::dirigir() {
        cout << "dirigir";
    }
}
  
```

Sobrescrita de método

- Métodos herdados podem ser sobrescritos
- Subclasse se comporta de maneira diferente para a mesma invocação do método
- Métodos sobrescritos substituem para a classe em questão os métodos da superclasse
- Superclasse continua com seu comportamento habitual, não sofrendo alterações

Sobrescrevendo métodos




```
class Mamifero {
    void Mamifero:andar() {
        cout << "andar (mamifero)";
    }
}

class Homem : public Mamifero {
    void Homem:andar() {
        cout << "andar (homem)";
    }
}

Homem *h = new Homem();
h->andar(); // andar(homem)

Mamifero *m = new Mamifero();
m->andar(); // andar (mamifero)
```

Impedindo a sobrescrita




- Operador **final** no arquivo cabeçalho (.h)
- Impede que o método em questão seja sobrescrito pelas subclasses
- Subclasses herdam o método e podem utilizá-lo normalmente, mas não alterá-lo

```
class Mamifero {
    void andar() final;
}

class Homem : public Mamifero {
    void Homem:andar() {
        cout << "andar (homem)";
    }
}
```

ERRO

Restringindo acesso



- Operador **protected**
- Restringe o acesso aos atributos e aos métodos definidos para somente as subclasses

```
class Mamifero {
    protected:
    void Mamifero:andar() {
        cout << "andar (mamifero)";
    }
}

class Homem : public Mamifero {
    public:
    void Homem:algumMetodo() {
        andar();
    }
}

Mamifero *m = new Mamifero();
m->andar(); // fatal error

Homem *h = new Homem();
h->algumMetodo(); // andar (mamifero)
```

Polimorfismo



- Capacidade de um mesmo método executar diferentes comportamentos, dependendo do objeto sobre o qual está sendo invocado
- Não possui operador pois é um conceito
- Situações mais comuns que geram polimorfismo:
 - Sobrescrita de métodos
 - Implementação de classes abstratas
 - Implementação de interfaces

Exemplo de polimorfismo



```

class Alugavel
{
public:
    // Não precisa implementar para esta classe
    virtual void alugar();
}

class Livro : public Alugavel
{
public:
    void Livro:alugar()
    {
        cout << "Alugando um Livro";
    }
}

class DVD : public Alugavel
{
public:
    void DVD:alugar()
    {
        cout << "Alugando um DVD";
    }
}

Livro *o1 = new Livro();
DVD *o2 = new DVD();

o1->alugar(); // Alugando um Livro
o2->alugar(); // Alugando um DVD
  
```

Exemplo de polimorfismo



```

class Gato
{
    void Gato:miar()
    {
        cout << "Miado de Gato";
    }
}

class GatoPersa : public Gato
{
    void GatoPersa:miar()
    {
        cout << "Miado de Gato Persa";
    }
}

Gato *o1 = new Gato();
GatoPersa *o2 = new GatoPersa();

o1->miar(); // Miado de Gato
o2->miar(); // Miado de Gato Persa
  
```

Herança Múltipla



- O C++ permite que uma classe herde de mais de uma classe mãe
- No geral em OO herança múltipla quebra o conceito
- Na prática não utilizaremos este recurso neste módulo, mas utilizaremos mais adiante no curso

```
class <nome> : public <nome1>, public <nome2>, ...
```

Aulas práticas e manuais on-line



Assista agora as aulas práticas.

[Clique aqui](#) para visualizar as aulas práticas disponíveis
