Algoritmos e Estruturas de Dados I
Aula 13.2 - Classes + Vetores, Polimorfismo e Tratamento de Exceção

Prof. Felipe Lara

Instanciando vários objetos de uma Classe

Código de exemplo:

```
#include <iostream>
#include <string>
using namespace std;
class Aluno {
        public:
            string nome;
            int idade;

            Aluno() {
                nome = "";
                idade = 0;
                }
            Aluno(string n, int i) : nome(n), idade(i) {}
};
```

```
int main() {
    Aluno a1("Ana", 20);
    Aluno a2("Bruno", 22);

    cout << a1.nome << endl;
    cout << a2.nome << endl;
}
```

2 Objetos instanciados

Como seria para 20 objetos?

Instanciando vários objetos de uma Classe

Código de exemplo:

```
#include <iostream>
#include <string>
using namespace std;
class Aluno {
        public:
            string nome;
            int idade;

            Aluno() {
                nome = "";
                idade = 0;
                }
            Aluno(string n, int i) : nome(n), idade(i) {}
};
```

```
int main() {
    Aluno a1("Ana", 20);
    Aluno a2("Bruno", 22);
            …
    Aluno a20("Felipe", 15);

    cout << a1.nome << endl;
    cout << a2.nome << endl;
}
```

20 Objetos instanciados

Não é indicado fazer assim!

Classe + Vetor (Opção 1)

Código de exemplo:

```
#include <iostream>
#include <string>
using namespace std;
class Aluno {
        public:
            string nome;
            int idade;

            Aluno() {
                nome = "";
                idade = 0;
                }
            Aluno(string n, int i) : nome(n), idade(i) {}
};
```

```
int main() {
   Aluno alunos[3] = {
        Aluno("Ana", 20),
        Aluno("Bruno", 22),
        Aluno("Carla", 19)
    };

}
```

Classe + Vetor (Opção 2)

```
#include <iostream>
using namespace std;
#include <string>
class Aluno {
        private:
            string nome;
            int idade;
        public:
                … //métodos construtores aqui
        void setDados(string n, int i) {
                nome = n;
                idade = i;
         }
        void imprime(){
                 cout << "Nome: " << nome << ", Idade: " << idade << “\n”;
            }
};
```

```
int main() {
    Aluno turma[3];  // vetor de 3 objetos

    turma[0].setDados("Ana", 20);
    turma[1].setDados("Bruno", 22);
    turma[2].setDados("Carla", 19);

    turma[0].imprime();
    turma[1].imprime();
    turma[2].imprime();
}
```

Classe + Vetor + For

```
… // todo o código aqui
int main() {
    Aluno turma[10];

    for (int i = 0; i < 10; i++) {
        string n;
        int idade;

        cout << "Digite o nome do aluno " << i + 1 << ": ";
        cin >> n; //lê até o primeiro espaço
        cout << "Idade: ";
        cin >> idade;

        turma[i].setDados(n, idade);
    }

    cout << "\n--- Lista de Alunos ---\n";
    for (int i = 0; i < 10; i++) {
        turma[i].imprime();
    }
}
```

Leitura de String usando getline

```
… // todo o código aqui
int main() {
    Aluno turma[10];

    for (int i = 0; i < 10; i++) {
        string n;
        int idade;

        cout << "Digite o nome do aluno " << i + 1 << ": ";
        getline(cin, n);  // lê até o '\n'  cin >> n;
        cout << "Idade: ";
        cin >> idade;

        turma[i].setDados(n, idade);
    }

    cout << "\n--- Lista de Alunos ---\n";
    for (int i = 0; i < 10; i++) {
        turma[i].imprime();
    }
}
```

PUC Minas

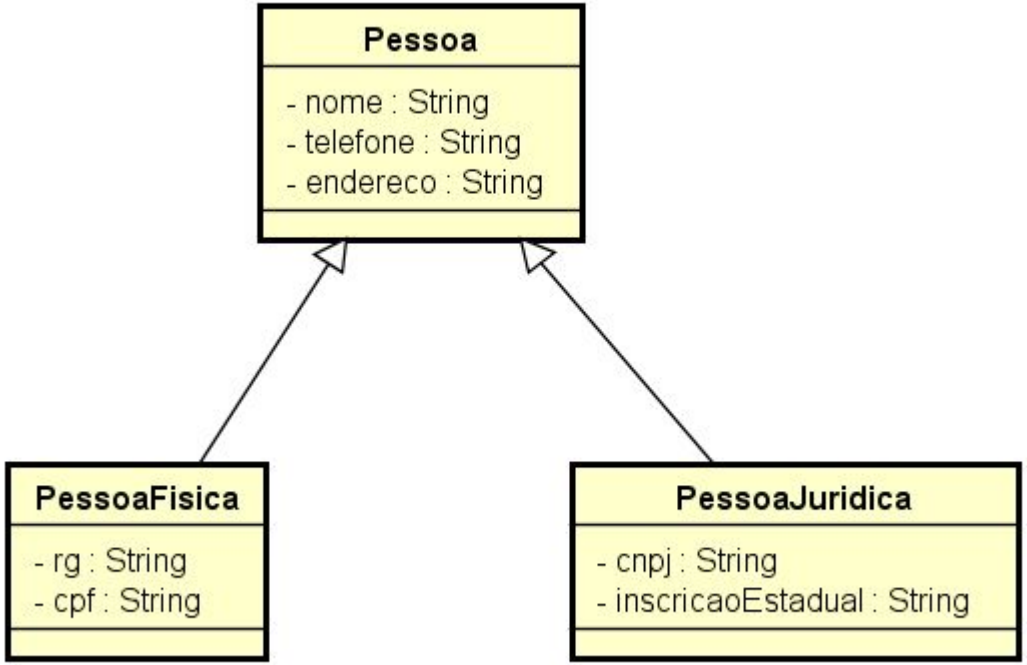
Polimorfismo

Polimorfismo

- Do grego poli + morphos = múltiplas formas
- Em termos de programação, significa que um **único nome de classe ou de método pode ser usado para representar comportamentos diferentes**.
- A decisão sobre qual comportamento utilizar é tomada em tempo de execução.
- O polimorfismo torna a programação orientada por objetos eficaz, permitindo a escrita de código genérico, fácil de manter e de estender.

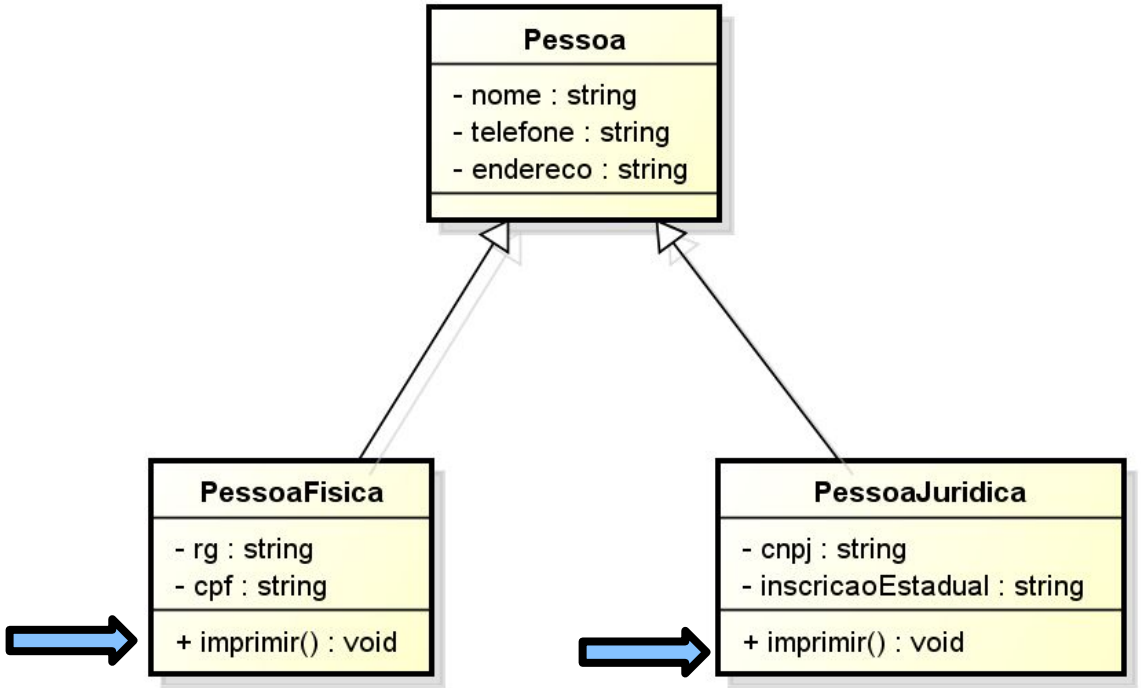
Polimorfismo

Suponha o mesmo exemplo anterior da hierarquia de Pessoa.



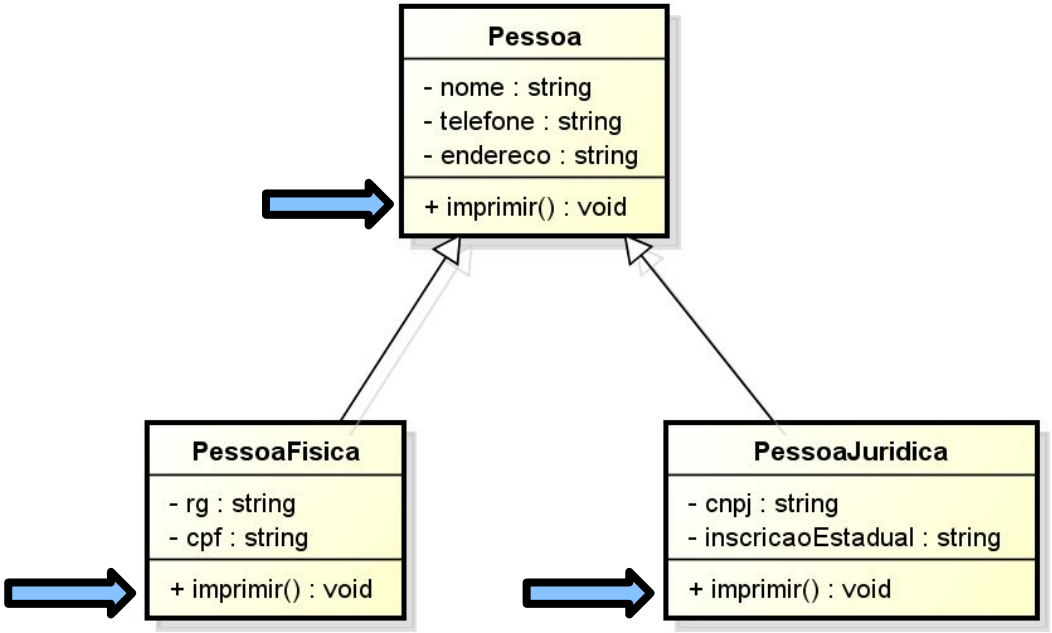
Polimorfismo

Vamos supor que queremos disponibilizar um método para imprimir uma PessoaFisica ou uma PessoaJuridica. Portanto:



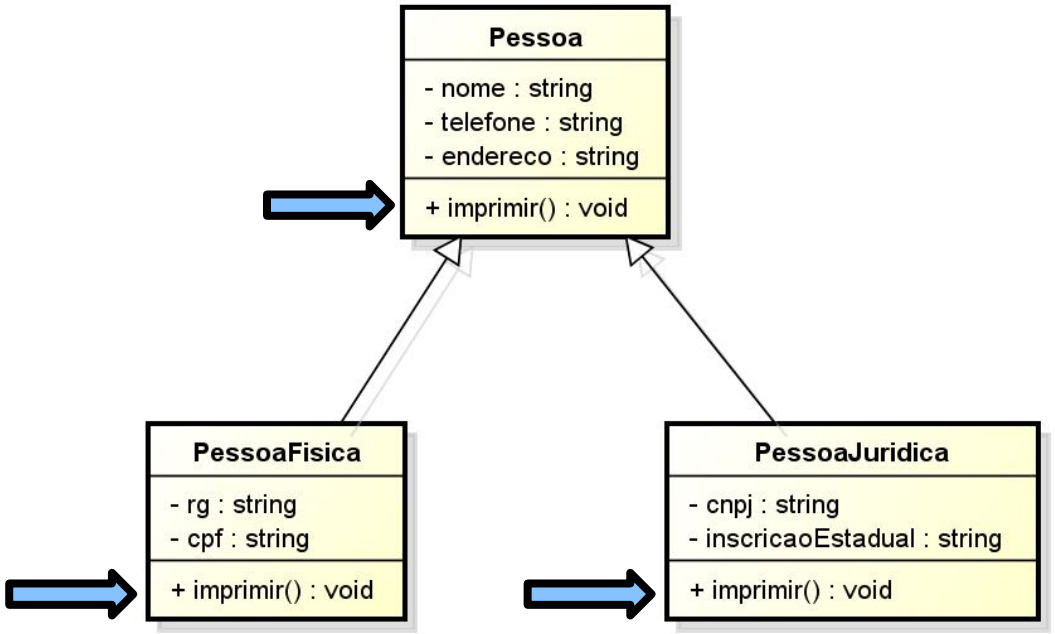
Polimorfismo

Podemos fazer melhor! Vamos disponibilizar também o método imprimir na superclasse Pessoa! Com isso, podemos imprimir uma pessoa sem precisar saber a sua natureza (física ou jurídica).



Polimorfismo

Podemos fazer melhor! Vamos disponibilizar também o método imprimir na superclasse Pessoa! Com isso, podemos imprimir uma pessoa sem precisar saber a sua natureza (física ou jurídica).



Isso é Polimorfismo!

- A pessoa ora irá se comportar como uma PessoaFisica, ora como uma PessoaJuridica.

- O polimorfismo ocorre em tempo de execução ao chamar o método imprimir sobre um ponteiro para Pessoa.

- O método imprimir() de Pessoa está sendo sobrescrito nas subclasses.

Exemplo 1 de Polimorfismo

```
#include <iostream>
#include <string>
using namespace std;

class Pessoa {
protected:
    string nome;
public:
    Pessoa(string nome){
        this->nome = nome;
    }
    virtual ~Pessoa() {}
    virtual void imprimir() {
        cout << "[Pessoa, nome=" << nome << "]" << endl;
    }
};
```

```
class PessoaFisica : public Pessoa {
    string cpf;

public:
    PessoaFisica(string nome, string cpf):
                            Pessoa(nome) {
        this->cpf = cpf;
    }

    void imprimir() override {
        cout << "[PessoaFisica, nome=" << nome <<
                ", cpf=" <<    cpf <<"]" << endl;
    }
};
```

Exemplo 1 de Polimorfismo

```
class PessoaJuridica : public Pessoa {
    string cnpj;

public:
    PessoaJuridica(string nome, string cnpj): Pessoa(nome) {
        this->cnpj = cnpj;
    }

    void imprimir() override {
        cout << "[PessoaJuridica, nome=" << nome
        cout << ", cnpj="      << cnpj <<"]" << “\n”;
    }
};
```

```
int main()
{
    PessoaFisica pessoaFisica("Joao da Silva",
"123456789-00");
    PessoaJuridica pessoaJuridica("Empresa do
Joao SA",    "12.299.535/0001-94");
    Pessoa* pessoa;
    pessoa = &pessoaFisica;
    pessoa->imprimir();
    pessoa = &pessoaJuridica;
    pessoa->imprimir();
    return 0;
}
```

Exemplo 2 de Polimorfismo

```
...
void imprimirPessoa(Pessoa* pessoa) {
    pessoa->imprimir();
}

int main()
{
    PessoaFisica pessoaFisica("Joao da Silva", "123456789-00");
    PessoaJuridica pessoaJuridica("Empresa do Joao SA",       "12.299.535/0001-94");
    imprimirPessoa(&pessoaFisica);
    imprimirPessoa(&pessoaJuridica);
    return 0;
}
```

PUC Minas

Tratamento de Exceção

O que é uma Exceção?

- Uma exceção é um evento indesejado ou inesperado que ocorre durante a execução do programa, rompendo o fluxo de execução normal das instruções do programa.

O que é uma Exceção?

- Uma exceção é um evento indesejado ou inesperado que ocorre durante a execução do programa, rompendo o fluxo de execução normal das instruções do programa.

- Por que tratar Exceções?
  - Se as exceções não forem tratados, **a aplicação é interrompida e encerrada**.

  - Seja qual for a causa da falha, espera-se que a aplicação seja capaz de detectar que houve uma situação anormal de funcionamento e gerenciar essa anormalidade de maneira adequada.

Tratamento de Exceção

Usamos tratamento de exceção com as cláusulas

- throw: lança uma exceção.
- try-catch: executa código que pode lançar exceção (try) e, caso seja lançada uma exceção, captura e trata (catch).

```
try {
  // protected code
}
catch( ExceptionName e1 ) {
  // catch block
}
catch( ExceptionName eN ) {
  // catch block
}
```

```
#include <iostream>
#include <exception>
using namespace std;

int main() {
        try {
            int x, y, z;
            cout << "Digite dois números inteiros: ";
            cin >> x >> y;
            if(y == 0) {
               throw exception();
            } else {
               z = x / y;
            }
            cout << "x/y = " << z << endl;
          }
        catch (exception& e) {
            cout << "excecao ocorreu: divisao por zero" << endl;
        }
           return 0;
}
```

```
#include <iostream>
#include <exception>
using namespace std;

int main() {
        try {
            int x, y, z;
            cout << "Digite dois números inteiros: ";
            cin >> x >> y;
            if(y == 0) {
               throw exception();
            } else {
               z = x / y;
            }
            cout << "x/y = " << z << endl;
          }
        catch (exception& e) {
           cout << "excecao ocorreu: divisao por zero" << endl;
        }
          return 0;
}
```

PUC Minas

Dúvidas?