|  |  |  |
| --- | --- | --- |
|  | **INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA MARANHÃO – CAMPUS BARRA DO CORDA** | |
| CURSO: Curso Técnico em Informática Integrado ao Ensino Médio | |
| DISCIPLINA: Programação Orientada a Objetos | DOCENTE: Reinaldo Antônio da Silva |
| DISCENTE: Victor Emanuel Sousa Lima | |
| TURMA: Informática II | DATA: 09/05/2021 |

Resumo

**PROGRAMAÇÃO ORIENTADA A OBJETOS**

Programação Orientada a Objetos (POO) é um paradigma da programação desenvolvido com o conceito de Objetos, que podem conter dados (ou atributos) organizados em campos, e métodos (ou procedimentos) equivalentes às funções na linguagem C.

A POO pode ser usada para automatizar diferentes situações. Digamos que em determinada situação exista várias **entidades**, que juntas formam um **domínio.** (Ex.: As entidades cliente, funcionário e conta podem formar um domínio bancário.) E ao trazemos para a OO, as entidades seriam os **objetos** e o domínio seria a **aplicação**. (Ex.: Para representar, em uma aplicação, o domínio bancário, seriam criados objetos para as entidades Cliente, Funcionário e Conta.)

Uma entidade (agora objeto) possui informações (Ex.: Um cliente tem nome, cpf, rg, data de nascimento, etc.), que são armazenadas em **atributos** do objeto. Uma entidade, também, faz ações (Ex.: Em uma conta existe a opção sacar, depositar e transferir dinheiro.), que são chamadas de **métodos** do objeto.

Um objeto é composto por atributos e métodos, logo temos que declará-los dentro de uma estrutura chamada de **Classe**, que servirá como um “molde” para a criação de objetos e para alocar o espaço necessário na memória ram. Sendo um objeto uma instância de uma classe (a "concretização" de uma classe)

Uma classe é declarada a partir da palavra reservada “**class**” e em seguida o seu nome (Ex.: class Cliente), que deve seguir o padrão “Pascal Case” (onde a primeira letra de qualquer palavra deve ser maiúscula. Assim como em nomes compostos, mas sem espaços entre as palavras.). Já os atributos são declarados primeiramente com o tipo e depois o nome. (Ex.: int idade;)

A partir de uma classe, pode-se criar quantos objetos forem necessários com a palavra reservada “new”. (Ex.: new Cliente();), que constrói o espaço na memória para o objeto.

 Para acessarmos os atributos e métodos de um objeto devemos ter uma variável de **referência** do objeto, que é declarada como um tipo não primitivo. (Ex.: Cliente Joao;), como o comando new devolve uma referência, podemos usá-los juntos. (Ex.: Cliente Joao = new Cliente;)

A manipulação de atributos se dá a partir do nome da referência, ponto(.) e o atributo. (Ex.: Joao.idade = 65;)

Caso não seja declarado um valor para os atributos, eles serão inicializados com valores padrões. Os atributos numéricos serão inicializados com o valor 0, os boolean serão inicializados com false e os demais serão inicializados com null (vazio).

Objetos de classes diferentes podem se relacionar a partir de **associações**.

Ex.:

class Cliente {

String nome;

String cpf;

String rg;

int idade;

}

class CartaoDeCredito {

String numero;

String dataDeVencimento;

Cliente cliente;

}

Os métodos são definidos como uma função.

Ex.:

class Conta {

double saldo;

Cliente cliente;

void tipoConta() {

System.out.println(“corrente”);

}

}

Para acessar um atributo dentro de métodos, usa-se a palavra reservada **this**.

Ex.:

void extrato() {

System.out.println(this.saldo);

}

Pode-se usar parâmetros nos métodos.

Ex.:

void deposita(double  valor) {

this.saldo += valor;

}

Pode-se, também, declarar métodos com nomes iguais, contanto que tenham parâmetros diferentes. Isso se denomina **sobrecarga**.

Ex.:

...

void compra(double  valor) {

this.saldo -= valor;

}

void compra(double  valor, float imposto) {

this.saldo -= (valor + (imposto/100 \* valor));

}

...

O que define se um método vai ser chamado são os parâmetros que são passados no ato.

Ex.:

...

Conta n1 = new Conta();

n1.compra(50.00)

n1.compra(100.00, 10)

...

Quando se quer inicializar atributos com valores não padrões, usa-se um método **construtor,** que deve possuir o mesmo nome da classe e parâmetros da mesma quantidade e tipo de atributo que se deseja inicializar.

Ex.:

class Cliente {

String nome;

String cpf;

String rg;

Cliente(String nome, String cpf, String rg) {

this.nome = nome;

this.cpf = cpf;

this.rg = rg;

}

}

Os construtores, também, podem ser sobrecarregados.

Ex.:

class Cliente {

String nome;

String cpf;

String rg;

Cliente(String nome, String cpf) {

this.nome = nome;

this.cpf = cpf;

}

Cliente(String nome, String cpf, String rg) {

this.nome = nome;

this.cpf = cpf;

this.rg = rg;

}

}

**ATRIBUTOS E MÉTODOS DE CLASSE**

Em determinada ocasião, pode-se necessitar de um atributo com um valor comum a todas as instâncias de uma classe (objetos), ou seja, um atributo da classe, e não da instância. Tal atributo seria um **atributo estático.**

Para tornar um atributo em estático, deve-se colocar a palavra reservada **static** antes de sua declaração.

Ex.:

class Conversor{

static double dolar = 5.24;

}

O mesmo conceito serve para **métodos estáticos.**

Ex.:

class Conversor{

static double dolar = 5.24;

static double retornaDolar(){

return Conversor.dolar;

}

}

Os atributos e métodos estáticos não precisam de uma instância para serem chamados, já que não são do objeto, e sim da classe.

Ex.:

…

// atributos estáticos

System.out.println(Conversor.dolar);

// métodos estáticos

Conversor.retornaDolar();

…

**ENCAPSULAMENTO**

Quando se quer definir se um atributo ou método pode ser acessado de qualquer parte de qualquer parte do sistema ou não, são usados os modificadores de acesso **public** e **private**.

public: Define que pode ser acessado de qualquer parte do sistema.

**Atributos públicos** podem ser usados e modificados de qualquer parte do sistema e **métodos públicos** podem ser chamados de qualquer parte do sistema.

private: Define que só pode ser acessado dentro da própria classe.

**Atributos privados** só podem ser acessados e modificados dentro da própria classe e **métodos privados** só podem ser chamados dentro da própria classe.

Podemos modificar o acesso dos atributos para privado, assim a aplicação ficaria mais segura e confiável. O acesso e a modificação dos atributos se daria por meio de métodos públicos, aumentando o grau de manutenibilidade. Trazemos assim, a ideia do **encapsulamento**.

Para a consulta e alteração dos valores de um atributo, usa-se os métodos Getters e Setters, respectivamente.

O método Getter retorna o valor de um atributo, já o método Setter recebe um parametro para modificar o valor de um atributo.

Ex.:

class Cliente {

private String nome ;

// Método Getter

public String getNome () {

return this.nome ;

}

// Método Setter

public void setNome ( String nome ) {

  this.nome = nome ;

}

}