PARADIGMAS DE PROGRAMAÇÃO

PARADIGMA PROCEDURAL

O paradigma procedural é um estilo de programação que se concentra na criação de procedimentos ou rotinas, onde cada procedimento é uma sequência de instruções que são executadas uma após a outra. Neste paradigma, o programa é dividido em partes menores chamadas de procedimentos, funções ou rotinas, e esses procedimentos são chamados sequencialmente para realizar uma tarefa específica.

Os principais conceitos do paradigma procedural são:

Procedimentos e Funções: No paradigma procedural, as tarefas são divididas em procedimentos ou funções. Um procedimento é uma sequência de instruções que realiza uma tarefa específica. Uma função é um tipo especial de procedimento que retorna um valor.

Variáveis e Escopo: As variáveis são usadas para armazenar valores temporários durante a execução do programa. No paradigma procedural, as variáveis têm escopo, o que significa que elas só são acessíveis dentro do procedimento ou função onde são declaradas, a menos que sejam passadas como parâmetros para outros procedimentos ou funções.

Controle de Fluxo: O controle de fluxo em um programa procedural é realizado principalmente por meio de estruturas de controle como condicionais (if, else, switch) e loops (for, while, do-while). Essas estruturas controlam a execução do programa com base em condições específicas.

Reutilização de Código: Uma das principais vantagens do paradigma procedural é a capacidade de reutilizar código. Procedimentos e funções podem ser chamados de diferentes partes do programa, permitindo que o mesmo código seja usado várias vezes.

Legibilidade e Manutenção: O paradigma procedural geralmente resulta em código mais fácil de entender e manter, especialmente para programas menores e médios. Isso ocorre porque o código é organizado em procedimentos separados, o que facilita a identificação e a correção de erros.

Exemplo de Programa Procedural em PHP:

```php
<?php

// Definição de uma função para calcular a área de um retângulo
function calcularAreaRetangulo($comprimento, $largura) {
    return $comprimento * $largura;
}

// Definição de uma função para imprimir uma mensagem
function imprimirMensagem($mensagem) {
    echo $mensagem;
}

// Chamada das funções
$area = calcularAreaRetangulo(5, 10);
imprimirMensagem("A área do retângulo é: " . $area);
?>
```

Embora o paradigma procedural tenha suas vantagens, ele também tem algumas limitações, especialmente em programas grandes e complexos. À medida que um programa cresce, ele pode se tornar difícil de manter devido à falta de encapsulamento e modularidade. Por esse motivo, em muitos casos, os desenvolvedores preferem usar paradigmas de programação mais modernos, como o paradigma orientado a objetos.

PARADIGMA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

O paradigma de programação orientada a objetos (POO) é um modelo de programação baseado no conceito de "objetos", que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos. Esses objetos são instâncias de classes, que atuam como modelos para os objetos. O POO é amplamente utilizado em diversas linguagens de programação, incluindo Java, Python, C++, C#, PHP, entre outras, e é caracterizado por alguns princípios fundamentais:

Os principais conceitos do paradigma orientado a objetos são:

Abstração: A abstração é a capacidade de representar objetos do mundo real em software. Em POO, uma classe é uma representação abstrata de um conceito do mundo real, e os objetos são instâncias dessa classe. Por exemplo, uma classe Carro pode representar todos os carros do mundo, enquanto um objeto específico dessa classe pode representar um carro em particular.

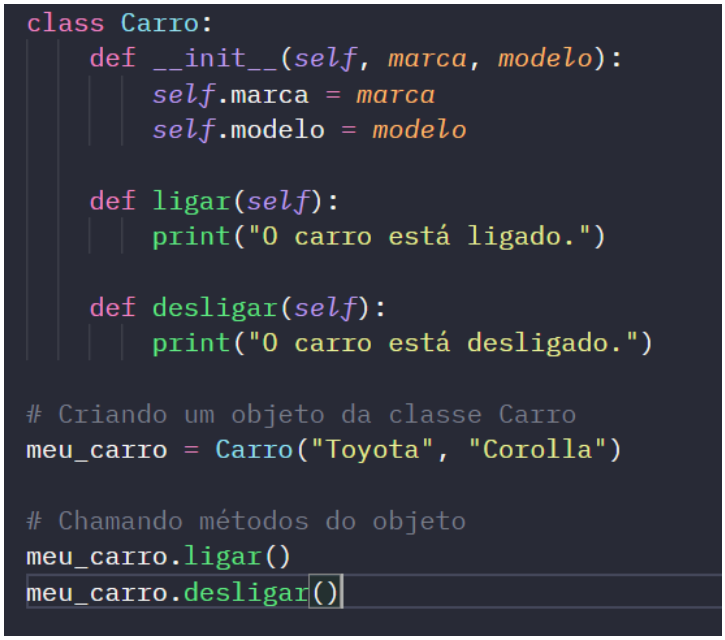
Encapsulamento: O encapsulamento é o conceito de ocultar os detalhes internos de um objeto e fornecer uma interface clara e consistente para interagir com ele. Isso é alcançado através do uso de modificadores de acesso, como public, private e protected, que controlam quais partes de uma classe são acessíveis a outros objetos.

Herança: A herança é um mecanismo que permite que uma classe (subclasse) herde atributos e métodos de outra classe (superclasse). Isso promove a reutilização de código e permite criar hierarquias de classes, onde classes mais específicas podem herdar comportamentos de classes mais genéricas.

Polimorfismo: O polimorfismo é a capacidade de um objeto ser tratado de várias maneiras, dependendo do contexto em que é usado. Isso permite que objetos de diferentes classes sejam tratados de forma uniforme, desde que compartilhem um conjunto comum de métodos ou comportamentos.

Classes e Objetos: Em POO, uma classe é um modelo ou planta baixa para criar objetos. Ela define os atributos e métodos que os objetos terão. Um objeto é uma instância de uma classe específica. Por exemplo, se temos uma classe Cachorro, um objeto dessa classe poderia ser um cachorro chamado "Rex".

Métodos e Atributos: Os métodos são funções associadas a uma classe que definem o comportamento dos objetos dessa classe. Os atributos são variáveis associadas a uma classe que armazenam o estado dos objetos.



```python
class Carro:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def ligar(self):
        print("O carro está ligado.")

    def desligar(self):
        print("O carro está desligado.")

# Criando um objeto da classe Carro
meu_carro = Carro("Toyota", "Corolla")

# Chamando métodos do objeto
meu_carro.ligar()
meu_carro.desligar()
```

COMPARAÇÃO ENTRE PARADIGMA PROCEDURAL E PARADIGMA ORIENTADO A OBJETOS NA PROGRAMAÇÃO

A programação é uma arte que se manifesta através de diferentes paradigmas, cada um com suas próprias características, vantagens e desvantagens. Dois dos paradigmas mais proeminentes são o paradigma procedural e o paradigma orientado a objetos. Enquanto o paradigma procedural se concentra em procedimentos e funções que manipulam dados, o paradigma orientado a objetos se baseia na criação de classes e objetos que interagem entre si. Vamos examinar as diferenças entre essas duas abordagens:

1. Organização do Código:

Procedural: No paradigma procedural, o código é organizado em torno de procedimentos ou funções. As funções são responsáveis por manipular os dados passados para elas como argumentos, resultando em um estilo de programação baseado em sequência de comandos.

Orientado a Objetos: Por outro lado, o paradigma orientado a objetos organiza o código em torno de classes e objetos. As classes são estruturas que definem os dados e os comportamentos dos objetos. Os objetos são instâncias dessas classes, capazes de interagir uns com os outros por meio de métodos e atributos.

2. Reutilização de Código:

Procedural: A reutilização de código no paradigma procedural geralmente é feita por meio da criação de funções que podem ser chamadas em diferentes partes do programa. No entanto, a reutilização é limitada pela necessidade de passar argumentos específicos para essas funções.

Orientado a Objetos: O paradigma orientado a objetos promove a reutilização de código por meio de herança e composição. A herança permite que uma classe herde características de outra, enquanto a composição permite que objetos contenham outros objetos como parte de sua estrutura.

3. Encapsulamento:

Procedural: No paradigma procedural, o encapsulamento é limitado, pois as funções podem acessar qualquer variável definida em seu escopo ou em escopos superiores. Isso pode levar a problemas de segurança e dificuldades de manutenção.

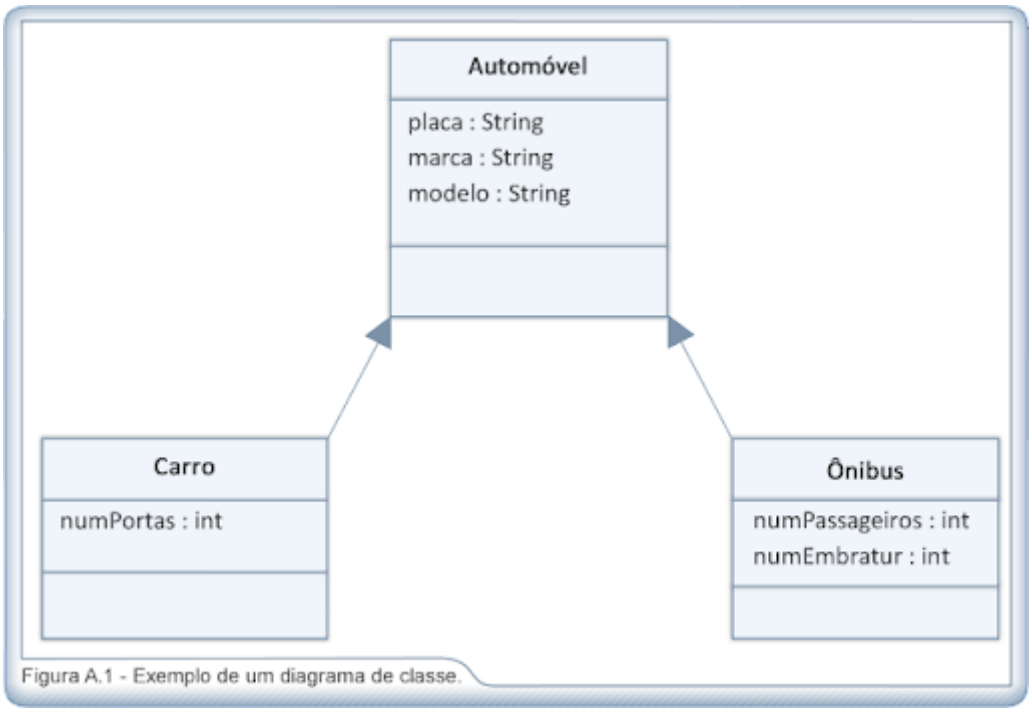
Orientado a Objetos: O encapsulamento é uma característica fundamental do paradigma orientado a objetos. Através do uso de modificadores de acesso, como public, private e protected, é possível controlar o acesso aos dados de uma classe, garantindo que eles sejam manipulados apenas de maneira segura e consistente.

4. Flexibilidade e Extensibilidade:

Procedural: O paradigma procedural pode ser menos flexível e extensível em comparação com o paradigma orientado a objetos. Adicionar novos recursos ou modificar o comportamento existente pode exigir mudanças em várias partes do código.

Orientado a Objetos: A orientação a objetos é conhecida por sua flexibilidade e extensibilidade. As classes podem ser facilmente estendidas ou modificadas sem afetar o código existente, tornando-o mais adaptável a mudanças e evolução.

DIAGRAMA DE CLASSE



Um diagrama de classe é uma representação visual da estrutura estática e das relações entre as classes de um sistema orientado a objetos. Ele descreve as classes, seus atributos, métodos e os relacionamentos entre as classes.

Principais finalidades do diagrama de Classe:

O diagrama de classe é uma das ferramentas mais fundamentais da UML e oferece várias vantagens no processo de desenvolvimento de software. Aqui estão algumas das principais vantagens:

1. **Visualização da estrutura estática:** O diagrama de classe fornece uma representação visual da estrutura estática de um sistema, mostrando as classes, seus atributos e métodos, bem como os relacionamentos entre elas. Isso ajuda os desenvolvedores a compreender melhor a arquitetura do sistema e as interações entre seus componentes.
2. **Comunicação eficaz:** Como uma ferramenta de modelagem visual, o diagrama de classe facilita a comunicação entre membros da equipe de desenvolvimento, clientes e outras partes interessadas. Ele oferece uma linguagem comum para discutir e documentar os requisitos do sistema, o design da solução e as decisões arquiteturais.

3. **Identificação de classes e objetos:** Ao criar um diagrama de classe, os desenvolvedores podem identificar as classes principais do sistema e seus atributos e métodos associados. Isso ajuda a definir claramente a estrutura do sistema e a determinar como os diferentes componentes devem interagir entre si.
4. **Análise de relacionamentos:** O diagrama de classe permite visualizar os relacionamentos entre as classes, como associações, agregações e composições. Isso é crucial para entender como os objetos se relacionam no sistema e para garantir que essas relações sejam modeladas de forma precisa e eficiente.
5. **Modelagem de herança e polimorfismo:** O diagrama de classe permite representar facilmente conceitos de herança e polimorfismo, mostrando como as classes derivadas herdam comportamentos e atributos das classes base. Isso ajuda a criar uma hierarquia de classes que promove a reutilização de código e a manutenção de um design coeso e flexível.
6. **Suporte a técnicas de desenvolvimento orientado a objetos:** O diagrama de classe é uma ferramenta essencial para o desenvolvimento orientado a objetos, pois permite modelar conceitos fundamentais como encapsulamento, abstração, herança e polimorfismo. Ele ajuda os desenvolvedores a aplicar essas técnicas de forma eficaz e a criar sistemas mais robustos e flexíveis.

Componentes de um Diagrama de Classe:

Classe: É representada por um retângulo dividido em três compartimentos. O primeiro compartimento contém o nome da classe, o segundo contém os atributos da classe e o terceiro contém os métodos da classe.

Atributos: São as características ou propriedades de uma classe. Eles são listados no segundo compartimento do retângulo que representa a classe. Cada atributo geralmente é acompanhado de seu tipo de dado.

Métodos: São as operações ou funções que uma classe pode realizar. Eles são listados no terceiro compartimento do retângulo que representa a classe. Cada método é acompanhado de sua lista de parâmetros e seu tipo de retorno, se houver.

Relacionamentos: Os relacionamentos entre as classes são representados por linhas que conectam as classes. Os principais tipos de relacionamentos são:

Associação: Indica que uma classe usa os serviços de outra classe. É representada por uma linha sólida.

Agregação: Indica que uma classe é composta por outras classes. É representada por uma linha com um losango vazio no extremo da classe composta.

Composição: É um tipo mais forte de agregação, onde a classe composta é parte integrante da classe que a contém. É representada por uma linha com um losango preenchido no extremo da classe composta.

Herança: Indica que uma classe herda atributos e métodos de outra classe. É representada por uma linha sólida com uma seta apontando para a classe pai.

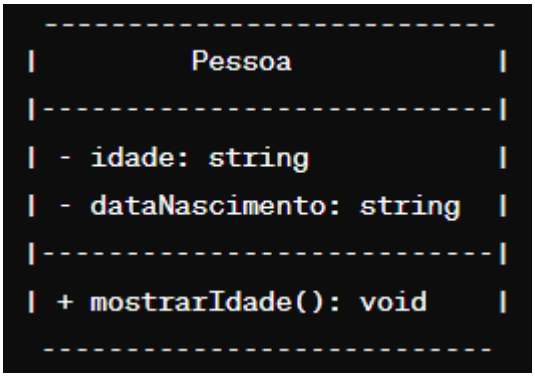
Implementação: Indica que uma classe implementa a interface definida por outra classe. É representada por uma linha pontilhada com uma seta apontando para a classe interface.

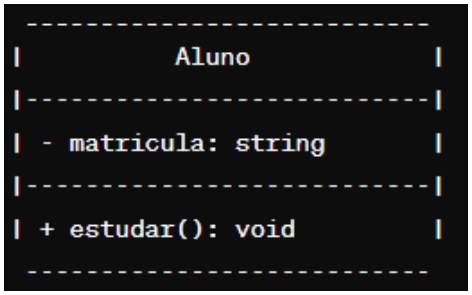
Código que implementa o diagrama acima:

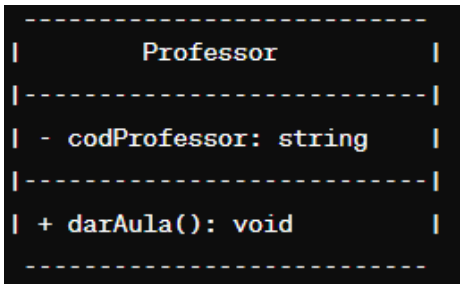
```php
<?php

class Automovel {

	protected $placa;
	protected $marca;
	protected $modelo
	public function __construct($placa,$marca,$modelo) {
	$this->placa = $placa;
	$this->marca = $marca;
	$this->modelo = $modelo
}

public function ligar(){
	echo "O automóvel está ligado.\n";
}

class Carro extends Automovel{
	protected $numPortas;


	public function __construct($placa, $marca, $modelo, $numPortas){
		parent::__construct($placa $marca, $modelo);
		$this->numPortas = $numPortas;
	}
}


class Onibus extends Automovel {
	protected $numPassageiros;
	public function __construct($placa, $marca, $modelo, $numPassageiros){
		parent::__construct($placa $marca, $modelo);
		$this->numPassageiros = $numPassageiros;
	}
}


// Exemplo de uso das classes


$carro = new Carro("ABC1234","Chevrolet","Onix", 4);
echo "Carro: Placa - {$carro->placa}, Número de Portas - {$carro->numPortas}\n";
$carro->ligar();

$onibus= new Onibus("XYZ5678","Mercedes","MB", 40)
echo "Número de Passageiros: - {$onibus->numPassageiros}\n";
$onibus->ligar();
```

EXERCÍCIO:

Escreva o diagrama de classe de uma classe chamada Pessoa com os atributos string:idade, string:dataNascimento e o método mostrarIdade(). Uma classe Aluno que herda de Pessoa e tem o atributo string:matricula e o método estudar(). Uma classe Professor que herda de Pessoa que tem o atributo codProfessor e o método darAula().








```php
class Pessoa {
    protected $idade;
    protected $dataNascimento;

    public function __construct($idade, $dataNascimento) {
        $this->idade = $idade;
        $this->dataNascimento = $dataNascimento;
    }

    public function mostrarIdade() {
        // Calcula a idade com base na data de nascimento
        $dataNascimento = new DateTime($this->dataNascimento);
        $hoje = new DateTime();
        $idade = $hoje->diff($dataNascimento)->y;

        echo "Idade: " . $idade . " anos";
    }
}
```

```php
class Aluno extends Pessoa {
    protected $matricula;

    public function __construct($idade, $dataNascimento, $matricula) {
        parent::__construct($idade, $dataNascimento);
        $this->matricula = $matricula;
    }

    public function estudar() {
        echo "Estudando...";
    }
}
```

```php
class Professor extends Pessoa {
    protected $codProfessor;

    public function __construct($idade, $dataNascimento, $codProfessor) {
        parent::__construct($idade, $dataNascimento);
        $this->codProfessor = $codProfessor;
    }

    public function darAula() {
        echo "Dando aula...";
    }
}
```