Residência em Tecnologia da Informação e Comunicação

POI - PRÁTICA ORIENTADA

INSTRUÇÃO PRÁTICA – PO-01

Marcelo da Silva Cruz

Pesquise no Google e responda às perguntas abaixo.

1. O que é uma classe em Java e qual é a diferença entre uma classe e um objeto? Dê 5 exemplos mostrando-os em C++ e em Java.

A diferença entre uma classe e um objeto é que a classe é a estrutura ou o plano de fundo que define um tipo de objeto, enquanto um objeto é uma instância específica dessa classe, criada a partir do plano de fundo definido pela classe. Um objeto é uma entidade real que possui atributos e comportamentos conforme definidos pela classe.

Exemplos em C++

Classe Pessoa em C++:

```
class Pessoa {
public:
    string nome;
    int idade;
    void apresentar() {
        cout << "Meu nome é " << nome << " e tenho " << idade << " anos." << endl;
    }
};
```

Objeto Pessoa em C++:

```
Pessoa pessoa1;
pessoa1.nome = "João";
pessoa1.idade = 25;
pessoa1.apresentar();
```

Classe Carro em C++:

```
class Carro {
public:
    string modelo;
    int ano;
    void acelerar() {
        cout << "Acelerando o carro " << modelo << " de " << ano << "." << endl;
    }
};
```

Objeto Carro em C++:

```
Carro carro1;
carro1.modelo = "Toyota";
carro1.ano = 2022;
carro1.acelerar();
```

Exemplos em JAVA

Classe Livro em JAVA:

```
public class Livro {
    public String titulo;
    public String autor;
    public void exibirInfo() {
        System.out.println("Livro: " + titulo + ", Autor: " + autor);
    }
}
```

Objeto Livro em JAVA:

```
Livro livro1 = new Livro();
livro1.titulo = "Aventuras de Java";
livro1.autor = "Carlos Código";
livro1.exibirInfo();
```

Classe Retângulo em JAVA:

```
public class Retangulo {
    public double comprimento;
    public double largura;
    public double calcularArea() {
        return comprimento * largura;
    }
}
```

Objeto Retângulo em JAVA:

```
Retangulo retangulo1 = new Retangulo();
retangulo1.comprimento = 5.0;
retangulo1.largura = 3.0;
double area = retangulo1.calcularArea();
System.out.println("Área do retângulo: " + area);
```

2. Como você declara uma variável em Java e quais são os tipos de dados primitivos mais comuns? Faça um paralelo entre isso e a mesma coisa na linguagem C++

Em Java, declara-se uma variável indicando o tipo de dado que ela irá armazenar, seguido pelo nome da variável. A sintaxe básica é:

Exemplo em JAVA:

```
tipoDeDado nomeDaVariavel;
```

Declaração de variáveis em Java:

```
int idade;          // Variável do tipo inteiro
double salario;     // Variável do tipo ponto flutuante (double)
boolean ativo;      // Variável do tipo booleano
String nome;        // Variável do tipo String (objeto que representa uma sequência de caracteres)
```

Tipos de dados primitivos mais comuns em Java:

a) int: Representa números inteiros.
b) Double: Representa números de ponto flutuante de precisão dupla.
c) Boolean: Representa valores verdadeiro ou falso.
d) Char: Representa um caractere único.
e) Byte, short, long, float: Outros tipos numéricos primitivos com diferentes faixas e precisões.

Exemplo em C++:

```
int idade;          // Variável do tipo inteiro
double salario;     // Variável do tipo ponto flutuante (double)
bool ativo;         // Variável do tipo booleano em C++
char letra;         // Variável do tipo caractere
std::string nome;   // Variável do tipo string em C++
```

Em C++, os tipos de dados primitivos são semelhantes:

a) int: Representa números inteiros.
b) Double: Representa números de ponto flutuante de precisão dupla.
c) Boolean: Representa valores verdadeiro ou falso.
d) Char: Representa um caractere único.
e) **(Não existe um tipo de dado primitivo string em C++):** Em C++, strings são tratadas como um tipo especial e são implementadas usando a classe “std::string” do namespace “std”.

Ambas as linguagens compartilham muitos conceitos em relação aos tipos de dados primitivos, embora haja algumas diferenças sutis na implementação e sintaxe. As semelhanças tornam mais fácil para os desenvolvedores aprenderem ambas as linguagens, uma vez que já estejam familiarizados com um desses paradigmas.

3. Explique o conceito de herança em Java e como você pode criar uma subclasse a partir de uma classe existente. Faça um paralelo com C++, apresentando 5 exemplos.

Herança em Java:

É um conceito fundamental na programação orientada a objetos que permite que uma classe (subclasse) herde atributos e métodos de outra classe (superclasse). A subclasse pode estender ou especializar a funcionalidade da superclasse, promovendo a reutilização de código e a criação de hierarquias de classes.

Para criar uma subclasse em Java, você usa a palavra-chave **extends**. Aqui está um exemplo simples:

```
// Superclasse
class Animal {
    void emitirSom() {
        System.out.println("Som genérico de um animal");
    }
}

// Subclasse
class Cachorro extends Animal {
    void latir() {
        System.out.println("Latindo...");
    }
}
```

Neste exemplo, a classe **Cachorro** herda da classe **Animal**. Agora, a classe **Cachorro** possui tanto o método **emitirSom** da classe **Animal** quanto o seu próprio método **latir**.

Herança em Java:

Em C++, o conceito de herança é semelhante, mas a sintaxe é um pouco diferente. A palavra-chave “**extends”** é substituída por “ **:** “ e você pode usar **“public”**, **“private”** e **“protected”** para definir a visibilidade dos membros da classe base na classe derivada. Aqui estão 5 exemplos em C++:

```
// Exemplo 1: Herança pública
class Animal {
public:
    void emitirSom() {
        cout << "Som genérico de um animal" << endl;
    }
};

class Cachorro : public Animal {
public:
    void latir() {
        cout << "Latindo..." << endl;
    }
};

// Exemplo 2: Herança privada
class Veiculo {
private:
    int velocidade;
public:
    void acelerar() {
        cout << "Acelerando..." << endl;
    }
};

class Carro : private Veiculo {
public:
    void dirigir() {
        acelerar(); // Ainda pode acessar membros públicos da classe base
        cout << "Dirigindo..." << endl;
    }
};

// Exemplo 3: Herança protegida
class Figura {
protected:
    double area;
public:
    void calcularArea() {
        cout << "Calculando área..." << endl;
    }
};

class Circulo : protected Figura {
```

```
public:
    void exibirArea() {
        calcularArea(); // Ainda pode acessar membros protegidos da classe base
        cout << "Área do círculo: " << area << endl;
    }
};

// Exemplo 4: Herança múltipla
class Pessoa {
public:
    void saudacao() {
        cout << "Olá, eu sou uma pessoa." << endl;
    }
};

class Programador {
public:
    void programar() {
        cout << "Escrevendo código..." << endl;
    }
};

class PessoaProgramadora : public Pessoa, public Programador {
};

// Exemplo 5: Herança virtual
class AnimalVirtual {
public:
    virtual void mover() {
        cout << "Movendo-se como um animal." << endl;
    }
};

class Peixe : public AnimalVirtual {
public:
    void mover() override {
        cout << "Nadando como um peixe." << endl;
    }
};
```

Esses exemplos mostram diferentes aspectos da herança em C++, incluindo herança pública, privada, protegida, múltipla e herança virtual. Cada exemplo ilustra como as classes derivadas podem herdar e usar membros da classe base.

4. Quando declaramos uma variável em Java, temos, na verdade, um ponteiro. Em C++ é diferente. Discorra sobre esse aspecto.

Em Java, a declaração de uma variável de um tipo de objeto cria uma referência a esse objeto, não um ponteiro direto para a área de memória onde o objeto está armazenado. Essa referência em Java é um identificador que aponta para o objeto no heap, mas o desenvolvedor não tem acesso direto à manipulação de ponteiros de memória.
Por exemplo, em Java:

```
// Declaração de uma variável de objeto
MinhaClasse objetoJava = new MinhaClasse();
```

Neste exemplo, **"objetoJava"** é uma referência à instância de **"MinhaClasse"** no heap, mas não é um ponteiro direto para o endereço de memória dessa instância.

Em C++, a situação é um pouco diferente. Quando você declara uma variável de um tipo de objeto, você está criando uma instância real desse objeto. A variável em C++ é, na verdade, um ponteiro quando você usa o operador **"new"** para alocar memória dinamicamente. No entanto, se você declara um objeto sem usar **"new"**, ele é alocado na pilha ou na área de dados estática, e você está manipulando diretamente o objeto.

Por exemplo, em C++:

```
// Declaração de uma variável de objeto (alocação estática)

MinhaClasse objetoCpp;

// Declaração de um ponteiro para um objeto (alocação dinâmica)

MinhaClasse* ponteiroCpp = new MinhaClasse();
```

No primeiro exemplo, **"objetoCpp"** é uma variável que representa diretamente uma instância de **"MinhaClasse"** alocada na pilha ou na área de dados estática. No segundo exemplo, **"ponteiroCpp"** é um ponteiro que aponta para uma instância de **"MinhaClasse"** alocada dinamicamente no heap.
Essa diferença fundamental na manipulação de objetos em Java e C++ reflete as características de cada linguagem. Java busca fornecer maior abstração e segurança de memória, enquanto C++ oferece maior controle sobre a alocação e liberação de memória, permitindo o uso direto de ponteiros.