Construtores, Destrutores, Encapsulamento e Métodos Get e Set em C++

1. Objetivos

- Compreender os conceitos de **métodos construtores e destrutores**.
- Explicar a criação de **construtores com e sem parâmetros**, incluindo **parâmetros com valores padrão**.
- Entender e aplicar o conceito de **encapsulamento**.
- Implementar e utilizar os **métodos get e set** para acessar e modificar atributos privados de uma classe.

---

2. Construtores em C++

2.1. O que é um Construtor?

Um **construtor** é um método especial que é automaticamente chamado quando um objeto é criado. Ele é usado para inicializar os atributos do objeto.

Principais características:

- Possui o mesmo nome da classe.
- Não possui tipo de retorno (nem `void`).
- Pode ter parâmetros (construtor parametrizado) ou não (construtor padrão).
- Pode ter **parâmetros com valores padrão**.
- É automaticamente chamado quando o objeto é criado.

Sintaxe Geral:

```
class NomeDaClasse {
public:
    NomeDaClasse() { // Construtor Padrão
        // Código de inicialização
    }
};
```

2.2. Construtor Padrão (Sem Parâmetros)

Este construtor é chamado automaticamente quando um objeto é criado **sem argumentos**.

Exemplo:

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

```cpp
class Pessoa {
public:
    Pessoa() { // Construtor padrão
        cout << "Objeto Pessoa criado!" << endl;
    }
};

int main() {
    Pessoa p1; // Chama automaticamente o construtor
    return 0;
}
```

Saída esperada:

```
Objeto Pessoa criado!
```

2.3. Construtor Parametrizado

Permite inicializar os atributos com valores personalizados durante a criação do objeto.

Exemplo:

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

class Pessoa {
public:
    string nome;
    int idade;

    Pessoa(string n, int i) { // Construtor parametrizado
        nome = n;
        idade = i;
    }
};

int main() {
    Pessoa p1("João", 25);
    cout << "Nome: " << p1.nome << ", Idade: " << p1.idade << endl;
    return 0;
}
```

Saída esperada:

```
Nome: João, Idade: 25
```

2.4. Construtor com Parâmetros de Valor Padrão

O C++ permite que parâmetros do construtor tenham valores padrão. Esses parâmetros devem ser definidos **da direita para a esquerda**.

Exemplo:

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

class Pessoa {
public:
    string nome;
    int idade;

    Pessoa(string n = "Desconhecido", int i = 0) { // Valores padrão
        nome = n;
        idade = i;
    }
};

int main() {
    Pessoa p1; // Usa os valores padrão
    Pessoa p2("Maria", 30);

    cout << "Nome: " << p1.nome << ", Idade: " << p1.idade << endl;
    cout << "Nome: " << p2.nome << ", Idade: " << p2.idade << endl;
    return 0;
}
```

Saída esperada:

```
Nome: Desconhecido, Idade: 0
Nome: Maria, Idade: 30
```

---

3. Destrutores em C++

3.1. O que é um Destrutor?

O **destrutor** é um método especial chamado automaticamente quando o objeto sai de escopo (ou é explicitamente destruído).

Principais características:

- Possui o mesmo nome da classe, mas precedido de um til (~).
- Não aceita parâmetros.
- Pode ser usado para liberar memória alocada dinamicamente.

Exemplo com liberação de memória:

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

class Pessoa {
public:
    int *idade;

    Pessoa(int i) {
        idade = new int(i);
    }
    ~Pessoa() {
        delete idade;
        cout << "Memória liberada!" << endl;
    }
};

int main() {
    Pessoa p1(25);
    return 0; // O destrutor é chamado automaticamente aqui
}
```

Saída esperada:

```
Memória liberada!
```

---

4. Encapsulamento em C++

4.1. O que é Encapsulamento?

O **encapsulamento** consiste em ocultar os detalhes internos de uma classe e permitir o acesso aos seus atributos por meio de **métodos de acesso (getters) e modificação (setters)**.

Por que usar o encapsulamento?

- **Controle de acesso**: Apenas os métodos públicos controlam o acesso aos atributos.
- **Validação de dados**: Os setters podem impor restrições sobre os valores atribuídos aos atributos.
- **Facilidade de manutenção**: Mudanças na estrutura interna da classe não afetam o código que usa a classe.

Modificadores de acesso:

- **public**: Acesso livre.
- **private**: Acesso apenas dentro da própria classe.
- **protected**: Acesso dentro da classe e das classes derivadas.

4.2. Exemplo prático de encapsulamento

Neste exemplo, criamos uma classe `Pessoa` onde os atributos **nome** e **idade** são privados, e o acesso a eles é feito por meio de **getters e setters**.

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

class pessoa {
private:
    string nome;
    int idade;

public:
    // Setter para o nome
    void set_nome(string n) {
        nome = n;
    }

    // Getter para o nome
    string get_nome() {
        return nome;
    }

    // Setter para a idade (validação para idade positiva)
    void set_idade(int i) {
        if (i > 0) idade = i;
    }

    // Getter para a idade
    int get_idade() {
        return idade;
    }
};

int main() {
    pessoa p;
    p.set_nome("João");
    p.set_idade(25);

    cout << "Nome: " << p.get_nome() << ", Idade: " << p.get_idade() << endl;
    return 0;
}
```

Saída esperada:

```
Nome: João, Idade: 25
```

4.3. Explicando os modificadores de acesso

1. `private`**:** Os atributos e métodos declarados como `private` só podem ser acessados por métodos da própria classe.

2. `public`**:** Permite o acesso ao método ou atributo de qualquer parte do programa.

3. `protected`**:** Permite o acesso a métodos e atributos para classes derivadas, mas impede o acesso de código externo.

4.4. Validação em Setters

Os **setters** podem incluir regras de validação antes de modificar o valor de um atributo. Por exemplo, podemos impedir que a idade de uma pessoa seja negativa.

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

class pessoa {
private:
    int idade;

public:
    void set_idade(int i) {
        if (i >= 0) {
            idade = i;
        } else {
            cout << "Idade inválida!" << endl;
        }
    }

    int get_idade() {
        return idade;
    }
};

int main() {
    pessoa p;
    p.set_idade(25); // Válido
    cout << "Idade: " << p.get_idade() << endl;

    p.set_idade(-5); // Inválido
    cout << "Idade: " << p.get_idade() << endl;

    return 0;
}
```

Saída esperada:

```
Idade: 25
Idade inválida!
Idade: 25
```

4.5. Exemplo de encapsulamento com classe mais completa

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

class conta_bancaria {
private:
    string titular;
    double saldo;

public:
    // Construtor parametrizado
    conta_bancaria(string nome_titular, double saldo_inicial) {
        titular = nome_titular;
        saldo = saldo_inicial > 0 ? saldo_inicial : 0; // Saldo inicial não pode
ser negativo
    }

    // Getters e Setters
    void set_titular(string nome) {
        titular = nome;
    }

    string get_titular() {
        return titular;
    }

    void set_saldo(double valor) {
        if (valor >= 0) {
            saldo = valor;
        } else {
            cout << "Valor inválido para saldo!" << endl;
        }
    }

    double get_saldo() {
        return saldo;
    }

    // Funções de depósito e saque
    void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
        } else {
            cout << "Depósito inválido!" << endl;
```

```
        }
    }

    void sacar(double valor) {
        if (valor > 0 && valor <= saldo) {
            saldo -= valor;
        } else {
            cout << "Saque inválido!" << endl;
        }
    }
};

int main() {
    conta_bancaria conta("João", 1000);
    conta.depositar(200);
    conta.sacar(500);
    conta.sacar(2000);

    cout << "Titular: " << conta.get_titular() << ", Saldo: R$ " <<
conta.get_saldo() << endl;
    return 0;

}
```

---

5. Métodos Get e Set

5.1. O que são Getters e Setters?

- **Getter**: Retorna o valor de um atributo privado.
- **Setter**: Define o valor de um atributo privado, garantindo que apenas valores válidos sejam atribuídos.

Exemplo prático:

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

class pessoa {
private:
    string nome;
    int idade;

public:
    void set_nome(string n) {
        nome = n;
    }
    string get_nome() {
        return nome;
    }

    void set_idade(int i) {
```

```
        if (i > 0) idade = i;
    }
    int get_idade() {
        return idade;
    }
};

int main() {
    pessoa p;
    p.set_nome("João");
    p.set_idade(25);

    cout << "Nome: " << p.get_nome() << ", Idade: " << p.get_idade() << endl;
    return 0;
}
```

Saída esperada:

```
Nome: João, Idade: 25
```
