Programação Orientada a objetos

CONSTRUTORES, DESTRUTORES E MODIFICADORES DE ACESSO

PROF. EDUARDO HABIB BECHELANE MAIA
HABIB@CEFETMG.BR

Construtores e Destrutores

Introdução

Construtores são as funções responsáveis por inicializar um objeto

- Qualquer inicialização necessária.

Implementador não precisa obrigatoriamente criar um construtor

- Compilador cria objetos vazios, nesse caso.
- Programador realiza a inicialização.

Ex.:

- Inicialização de conexões
- Inicialização de atributos
- Etc.

Exemplo Construtor

```
#include <iostream>

using namespace std;

class Circulo {
public:
	int pos_X, pos_Y;
	int raio;
	Circulo ( int X, int Y, int r ) { //Construtor
		pos_X = X;
		pos_Y = Y;
		raio = r;
	}

	void imprimirDadosCirculo()
	{
		cout << "x = " << pos_X << endl;
		cout << "y = " << pos_Y << endl;
		cout << "r = " << raio << endl;
	}
};
int main () { //Instanciando um objeto usando o Construtor
	Circulo c1 ( 10, 10, 5 );
	c1.imprimirDadosCirculo();

	system ("pause");
}
```

Construtor em Java

```
public class Circulo {
   public int pos_X, pos_Y, raio;

   // Construtor
   public Circulo(int X, int Y, int r) {
      pos_X = X;
      pos_Y = Y;
      raio = r;
   }

   public void imprimirDadosCirculo() {
      System.out.println("x = " + pos_X);
      System.out.println("y = " + pos_Y);
      System.out.println("r = " + raio);
   }

   public static void main(String[] args) {
      // Instanciando um objeto usando o Construtor
      Circulo c1 = new Circulo(10, 10, 5);
      c1.imprimirDadosCirculo();
   }
}
```

Destrutores

Os Destrutores, ao contrário dos construtores, são executados no momento da finalização do objeto.

- Liberar memória
- Finalizar Conexão

Destrutores

- Chamado quando um objeto da classe sai do escopo ou é explicitamente destruído
- Tem o mesmo nome da classe
  - Em c++ mas precedido por um til (~)
  - Chamado automaticamente quando o objeto sai do escopo ou quando delete é usado
  - Útil para:
    - liberar recursos como memória alocada dinamicamente
    - fechar arquivos
    - desconectar de um banco de dados, etc.

Destrutores

```
class Matriz {
        public:
                unsigned int lin, col;
                int ** mat; //Ponteiro == Vai precisar de al. dinamica
                Matriz ( unsigned int l, unsigned int c ) {
                        lin = l; col = c;
                        mat = new int* [l];
                        for ( unsigned int i = 0; i < l; i++ )
                        mat[i] = new int[col];
                }
                virtual ~Matriz() {
                        for (int i=0; i<col; i++ )
                                delete mat[i];
                        delete mat;
                }
};
```

Exemplo 2

```
#include <iostream>
#include <fstream>

class FileHandler {
private:
    std::fstream file;

public:
    // Construtor: abre o arquivo
    FileHandler(const std::string& filename) {
        file.open(filename, std::ios::in | std::ios::out);
        if (file.is_open()) {
            std::cout << "Arquivo"<<filename << "aberto\n“;
        } else {
            std::cout << "Falhou:" << filename << “\n“;
        }
    }

    // Destrutor: fecha o arquivo
    ~FileHandler() {
        if (file.is_open()) {
            file.close();
            std::cout << "Arquivo fechado" << std::endl;
        }
    }
    // Outras funções para manipular o arquivo (ler, escrever, etc.)
};

int main() {
    {
        FileHandler handler("exemplo.txt");
        // Operações com o arquivo...

    } // O destrutor é chamado aqui, fechando o arquivo.

    std::cout << "Fim do programa." << std::endl;

    return 0;
}
```

Destrutores em Java

Em Java, a linguagem tem um coletor de lixo

- Chamada quando um objeto deixa de ser usado
  - Ex: obj = null
- Para fazer o mesmo do exemplo do FileHandles, em c++, recomenda-se utilizar o método close()
- Em java, entretanto, existe o finally, que faz o mesmo que um destrutor

Em Java

```
import java.io.*;
public class FileHandler implements AutoCloseable {
    private BufferedReader reader;
    private BufferedWriter writer;

    // Construtor: abre o arquivo
    public FileHandler(String filename) {
        try {
            reader = new BufferedReader(new
FileReader(filename));
            writer = new BufferedWriter(new
FileWriter(filename, true)); // Abre para leitura e
escrita
            System.out.println("Arquivo " + filename);
        } catch (IOException e) {
            System.out.println("Falha ao abrir”);
        }
    }

    // Método close: fecha o arquivo
    @Override
    public void close() {
        try {
            if (reader != null) {
                reader.close();
            }
            if (writer != null) {
                writer.close();
            }
            System.out.println("Arquivo fechado.");
        } catch (IOException e) {
            System.out.println("Falha ao fechar.");
        }
    }

    // Outras funções para manipular o arquivo)
}

public class Main {
    public static void main(String[] args) {
        try (FileHandler handler = new
FileHandler("exemplo.txt")) {
            // Operações com o arquivo...
        } catch (Exception e) {
            System.out.println("Um erro ocorreu.");
        }

        System.out.println("Fim do programa.");
    }
}
```

Exemplo com finalize

```
import java.io.*;
public class FileHandler {
    private BufferedReader reader;
    private BufferedWriter writer;

    // Construtor: abre o arquivo
    public FileHandler(String filename) {
        try {
            reader = new BufferedReader(new
FileReader(filename));
            writer = new BufferedWriter(new
FileWriter(filename, true)); // Abre
            System.out.println("Arquivo aberto.");
        } catch (IOException e) {
            System.out.println("Falha ao abrir.");
        }
    }

    @Override
    protected void finalize() throws Throwable {
        try {
            if (reader != null) {
                reader.close();
            }
            if (writer != null) {
                writer.close();
            }
            System.out.println("Arquivo fechado.");
        } catch (IOException e) {
            System.out.println("Falha ao fechar.");
        } finally {
            super.finalize();
        }
    }

    // Outras funções para manipular o arquivo
}

public class Main {
    public static void main(String[] args) {
        FileHandler handler = new
FileHandler("exemplo.txt");
        // Operações com o arquivo...
        // ...
     /*Tenta forçar a coleta de lixo, mas não há   garantia
de que finalize() será chamado*/
        handler = null;
        System.gc();
        System.out.println("Fim do programa.");
    }
}
```

Proteção de dados

MODIFICADORES DE ACESSO

Introdução

Muitas vezes não queremos que as outras classes tenham acesso direto a algum componente da classe

- Atributos
- Métodos
- etc;.

Introdução

Um dos fundamentos da orientação à objetos é evitar que classes tenham acesso a um código que não tenha a ver com sua lógica.

É possível realizar o controle de acesso aos elementos

- Efetiva implementação do encapsulamento.

Pode-se controlar esse acesso com os “modificadores de acesso".

Conhecidos pelos identicadores "public", "private" e "protected".

Modificadores de acesso

Modificam os direitos de acesso que as classes e funções externas têm sobre os elementos de uma classe.

- **private:** Acessível apenas dentro da mesma classe.
- **protected:** Acessível dentro da mesma classe e classes derivadas (em java, acessível em classes do mesmo pacote também)
- **public:** Acessível de qualquer lugar onde o objeto da classe é visível.
- **default (só em java):** Acessível dentro da mesma classe e outras classes no mesmo pacote. Não tem equivalente direto em C++.

Modificadores de acesso

Em c++ quando não é colocado nenhum modificador de acesso, os membros serão, obrigatoriamente, do tipo "protected".

Em java, serão default

Se não há heranca, não há diferença entre "private" e "protected“ em c++.

Exemplo 1

Imagine uma classe Corpo, que possui massa, volume e consequentemente densidade.

- Se alterarmos a massa de um corpo sem alterarmos a densidade, criamos uma inconsistência dentro da classe.
- Por esse exemplo vemos por que é importante controlar o acesso aos atributos da classe.

Exemplo 1

```
// Corpo.hpp
#pragma once  // Para garantir que o header só seja incluído uma vez durante a compilação

class Corpo {
private:
    float massa;                                    //O modificador private impede que qualquer código externo à própria
    float densidade;                                //classe Corpo tenha acesso aos campos de Corpo
    float volume;

public:
    // Construtor
    Corpo(float massa, float volume);               //O construtor vai definir o valor da densidade logo na criação do objeto.

    // Métodos getters
    float getMassa() const;
    float getDensidade() const;
    float getVolume() const;

    // Métodos setters
    void setMassa(float massa);
    void setVolume(float volume);
    void atualizarDensidade();
};
```

Exemplo 1

(Acesso através de métodos públicos)

```
// Corpo.cpp
#include "Corpo.hpp"

// Construtor
Corpo::Corpo(float massa, float volume) {
    this->massa = massa;
    this->volume = volume;
    this->densidade = massa / volume;
}

// Métodos getters
float Corpo::getMassa() const {
    return massa;
}

float Corpo::getDensidade() const {
    return densidade;
}

float Corpo::getVolume() const {
    return volume;
}
```

```
// Métodos setters
void Corpo::setMassa(float massa) {
    this->massa = massa;
    atualizarDdensidade();
}

void Corpo::setVolume(float volume) {
    this->volume = volume;
    atualizarDdensidade();
}

// Método para atualizar a densidade sempre
que a massa ou o volume são alterados
void Corpo::atualizarDensidade() {
    if (volume != 0) {  // Evita divisão por zero
        this->densidade = massa / volume;
    }
}
```

Em Java

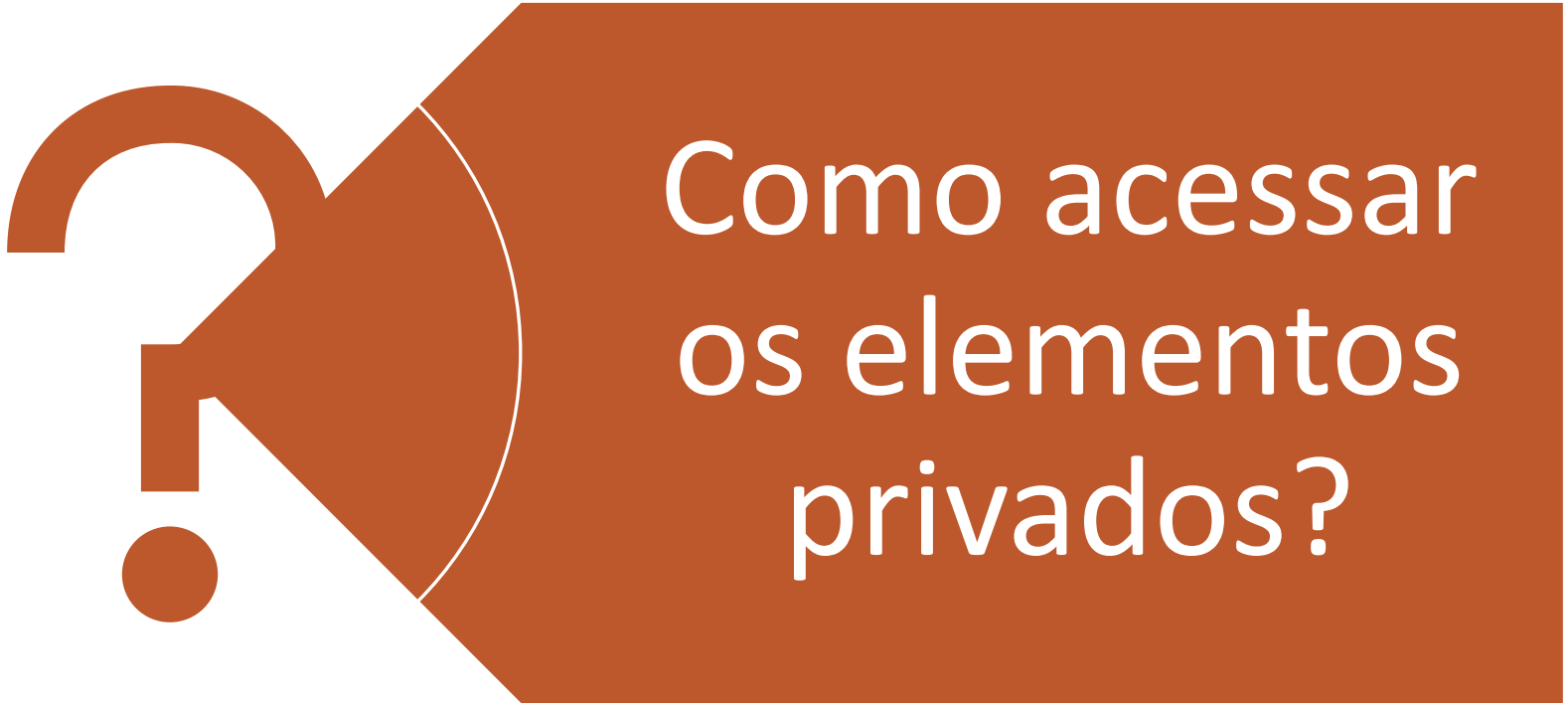
Classe Corpo em Java

```
public class Corpo {
    private float massa;
    private float densidade;
    private float volume;

    // Construtor
    public Corpo(float massa, float volume) {
        this.massa = massa;
        this.volume = volume;
        atualizarDdensidade();
    }

    // Métodos getters
    public float getMassa() {
        return massa;
    }

    public float getDensidade() {
        return densidade;
    }

    public float getVolume() {
        return volume;
    }

    // Métodos setters
    public void setMassa(float massa) {
        this.massa = massa;
        atualizarDensidade();
    }

    public void setVolume(float volume) {
        this.volume = volume;
        atualizarDensidade();
    }

    // Método para atualizar a densidade
    private void atualizarDensidade() {
        if (volume != 0) {  // Evita divisão por zero
            this.densidade = massa / volume;
        }
    }
}
```

Main.java

```
public class MainApp {
    public static void main(String[] args) {
        Corpo corpo = new Corpo(10, 2);
        System.out.println("Massa: " + corpo.getMassa());
        System.out.println("Volume: " + corpo.getVolume());
        System.out.println("Densidade: " + corpo.getDensidade());

        corpo.setMassa(20);
        System.out.println("Nova Massa: " + corpo.getMassa());
        System.out.println("Nova Densidade: " + corpo.getDensidade());
    }
}
```

Pergunta



Formas de inicializar um construtor em c++

Primeira forma

```
ContaBancaria::ContaBancaria(int agencia, int cnt, float s) : agencia(agencia), conta(cnt), saldo(s) {}
```

Segunda forma

```
ContaBancaria::ContaBancaria(int agencia, int cnt, float s) {
   this->agencia = agencia;
   this->conta = cnt;
   this->saldo = s;
}
```

Primeira forma: Mais eficiente, pois inicializa os membros diretamente quando o objeto é criado.

Segunda Forma: Menos eficiente porque primeiro os membros são construídos e depois são atribuídos.

Exemplo 2

ContaBancaria.hpp

```
#ifndef __CONTA_BANCARIA_HPP
#define __CONTA_BANCARIA_HPP
#include <string>
class ContaBancaria {
        private:
            int agencia;
            int conta;
            float saldo;

        public:
            ContaBancaria(int agencia, int cnt, float s);
            ContaBancaria(int agencia, int cnt);
            ContaBancaria();

            int getAgencia();
            void setAgencia(int a);

            int getContaBancaria();
            void setContaBancaria(int c);

            float getSaldo();
            void setSaldo(float saldo);

            void saque(float valor);
            void deposito (float valor);
};
#endif
```

Exemplo 2

Se tentar acessar algum membro não público a partir do main, por exemplo, uma mensagem de erro será exibida:

```
int main(){
    ContaBancaria cb;
    cb.agencia=1;
}
```

Ao compilar gerou o seguinte erro:

```
[Error] 'int ContaBancaria::agencia' is private
[Error] within this context
```

Exemplo 2

Em c++:

- https://onlinegdb.com/32k9NxB1s

Em java:

- https://onlinegdb.com/6W-D8ko0o

Proteção de dados

	private	protected	public	Default (Java)
Mesma classe	sim	sim	Sim	sim
Subclasses	não	sim	sim	não
Classes no mesmo pacote (Java)	não	sim	sim	sim
Classes diferentes	não	não	sim	não

Proteção de dados

Todos os **atributos** e **operações** de uma classe podem ser acessados pelas operações da mesma classe;

O acesso aos atributos é, em geral, **privado** ou **protegido**;

O acesso às operações que fazem parte da interface da classe é, em geral, **público**.

Atributos Estáticos

Static

Podemos distinguir variáveis em dois tipos:

- De instância; cada objeto possui uma cópia distinta desta variável
- De classe, só existe uma por classe.

Qual a palavra reservada para fazer essa distinção?

- Static

Atributo estático e não estático

Um **atributo estático** é um atributo cujo valor é compartilhado por todas as instâncias, objetos, da classe;

O acesso à um atributo estático é independente de objeto.

Um **atributo não estático** possui um valor único para cada objeto, instância da classe;

O acesso à um atributo não estático é dependente de objeto.

Exemplo de Pais

Sem Lista

- Em c++:
  - https://onlinegdb.com/QtF__Ifp8
- Em java:
  - https://onlinegdb.com/vSeW-7qh-

Com Lista

- Em c++:
  - https://onlinegdb.com/mtOUqnaJnM
- Em Java:
  - https://onlinegdb.com/VNGBpbu3x
  - Usando ArrayList:
    - https://onlinegdb.com/euGe0QWE8