

Programação Orientada a Objetos

**DESTRUTORES**

*em*

**C++**

# Introdução

- Os **construtores** fazem a **inicialização** de objetos
  - São funções membro especiais
  - Chamados **automaticamente**

Jogo	Memória		
	nome	""	0xCB2B = gears
	preco	0	0xCB2F
	horas	0	0xCB33
	custo	0	0xCB37

```
// criação do objeto  
Jogo gears;
```

```
Jogo::Jogo()  
{  
    // inicializam atributos  
    nome = "";  
    preco = 0;  
    horas = 0;  
    custo = 0;  
}
```

# Introdução

- É **recomendado** sempre **fornecer um construtor**
  - Para inicializar todos os atributos
  - Senão o compilador cria um construtor vazio
  - Uma classe pode ter **mais de um construtor**

```
Jogo::Jogo()  
{  
    nome  = "";  
    preco = 0;  
    horas = 0;  
    custo = 0;  
}
```

```
Jogo::Jogo(const string & titulo, float valor)  
{  
    nome  = titulo;  
    preco = valor;  
    horas = 0;  
    custo = valor;  
}
```

# Destrutores

- C++ possui outra **função membro especial**
  - Chamada de **destrutor**
  - Tem o mesmo nome da classe precedido por um til
  - Não possui parâmetros
  - Não possui retorno

```
Jogo::~~Jogo()  
{  
  
}
```

Ao contrário do construtor, o **destrutor** não recebe valores.

# Destrutores

- Adicionando o destrutor na classe Jogo
  - Temos uma classe completa

```
class Jogo
{
private:
    string nome;
    float preco;
    int horas;
    float custo;

public:
    Jogo(const string & titulo, float valor = 0);
    ~Jogo();
    void atualizar(float valor);
    void exibir();
    void jogar(int tempo);
};
```

```
Jogo::Jogo(
    const string & titulo,
    float valor)
{
    nome = titulo;
    preco = valor;
    horas = 0;
    custo = valor;
}

Jogo::~~Jogo()
{
    // vazio
}
```

# Destrutores

- O destrutor é **chamado**:
  - Automaticamente
  - Quando a **vida do objeto** chega ao fim

```
void processar()  
{  
    // construtor chamado  
    Jogo gears;  
    ...  
} // destrutor chamado
```

Variáveis locais são  
liberadas da memória  
ao **final do bloco**

# Destrutores

- A função do destrutor é **encerrar/destruir** coisas
  - Importante para se trabalhar com recursos
    - **Alocação dinâmica de memória**
    - Leitura de arquivos
    - Abertura de conexões

```
Conjunto::Conjunto(int n)
{
    // aloca memória
    vet = new int[n];
}
```

```
Conjunto::~~Conjunto()
{
    // libera memória
    delete [] vet;
}
```

# Destrutores

- O **construtor** da **classe Jogo** não aloca recursos
  - O seu **destrutor** pode ser vazio

```
Jogo::Jogo()
{
    // atributos

    nome = "";
    preco = 0;
    horas = 0;
    custo = 0;
}
```

```
Jogo::~~Jogo()
{
    // os atributos
    // da classe são
    // destruídos com
    // o objeto
}
```

	Memória	
Jogo	nome	"" 0xCB2B = gears
	preco	0 0xCB2F
	horas	0 0xCB33
	custo	0 0xCB37

A destruição do objeto  
não requer a **liberação**  
de recursos



# Destrutor Padrão

- E se a classe **não definir um destrutor?**
  - O compilador cria um destrutor padrão

```
class Jogo
{
private:
    string nome;
    float preco;
    int horas;
    float custo;

public:
    Jogo(const string & titulo, float valor = 0);
    void atualizar(float valor);
    void exibir();
    void jogar(int tempo);
};
```

Destrutor criado automaticamente pelo **compilador**

```
// destrutor padrão
Jogo::~~Jogo()
{
}
```

# Ciclo de Vida

- Quando a vida de um objeto chega ao fim?

```
// criação do objeto  
Jogo gears;
```

- Depende de como e onde o objeto foi criado:
  - Variável estática ou global
  - Variável local ou parâmetro
  - Alocado dinamicamente
  - Temporário

# Ciclo de Vida

## ▪ Objeto **global**

Variável Global	
Criação	Destruição
início do programa	fim do programa

Jogo	Memória	
	nome	"Gears" 0xCB2B = gears
	preco	0 0xCB2F
	horas	0 0xCB33
	custo	0 0xCB37



Construtor

```
// variável global  
Jogo gears { "Gears" };
```

```
int main()  
{  
    ...  
}
```



Destrutor

# Ciclo de Vida

## ▪ Objeto **estático**

Variável Estática	
Criação	Destruição
declaração da variável	fim do programa

Jogo	Memória	
	nome	0xCB2B = doom
	preco	0xCB2F
	horas	0xCB33
	custo	0xCB37

```
void processar()
{
    // variável estática
    static Jogo doom { "Doom" };
    ...
}
```

Construtor

```
int main()
{
    // chamada de função
    processar();
    ...
}
```

Destruitor

# Ciclo de Vida

## ▪ Objeto local

Variável Local	
Criação	Destruição
declaração da variável	fim do bloco

Jogo	Memória	
	nome	"GTA" 0xCB2B = gta
	preco	0 0xCB2F
	horas	0 0xCB33
	custo	0 0xCB37

```
// parâmetro de função
void processar(Jogo j) •----- Construtor
{
    ...
} •----- Destrutor

int main()
{
    // variável local
    Jogo gta { "GTA" }; •----- Construtor

    // chamada de função
    processar(gta);
    ...
} •----- Destrutor
```

# Ciclo de Vida

## ▪ Objeto **alocado dinamicamente**

Alocação Dinâmica	
Criação	Destruição
Execução do new	Execução do delete

Jogo	Memória	
	nome	"RDR" 0xCB2B = rdr
	preco	0 0xCB2F
	horas	0 0xCB33
	custo	0 0xCB37

```
int main()
{
    // alocação dinâmica
    Jogo * rdr = new Jogo { "RDR" }; •----- Construtor

    // chamada de função
    processar(rdr);
    ...
}

// parâmetro de função
void processar(Jogo * pJ)
{
    delete pJ; •----- Destrutor
}
```

# Ciclo de Vida

- Podemos **acompanhar o ciclo de vida** de um **objeto**
  - Usando seu construtor e destrutor

```
class Jogo
{
private:
    string nome;
    float preco;
    int horas;
    float custo;

public:
    Jogo();
    ~Jogo();
    ...
};
```

```
Jogo::Jogo()
{
    cout << "Construindo objeto\n";
    nome = "";
    preco = custo = 0.0f;
    horas = 0;
}

Jogo::~~Jogo()
{
    cout << "Destruindo objeto\n";
}
```

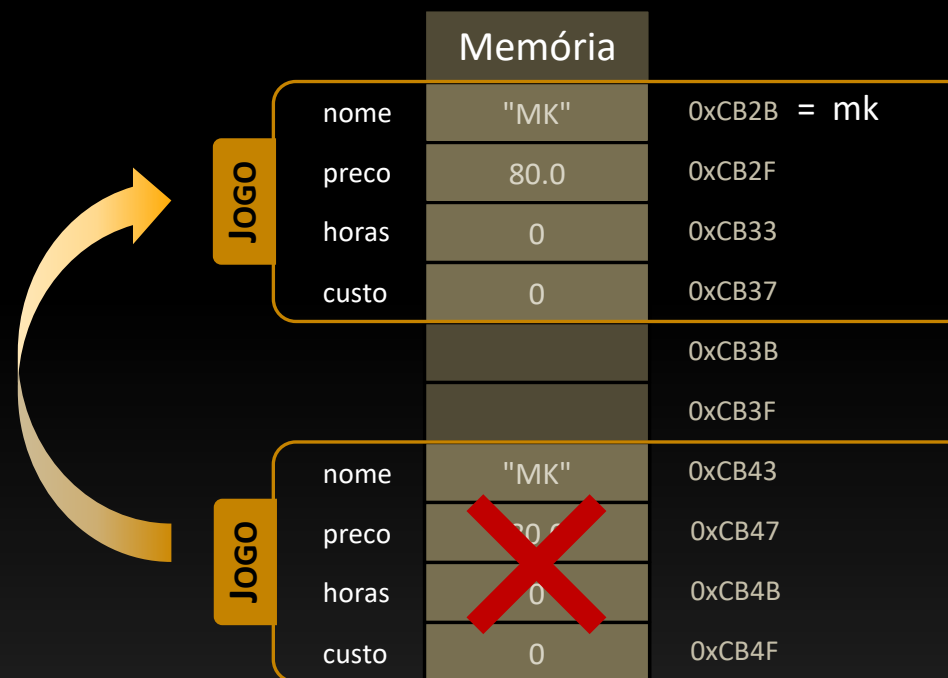
# Temporários

- **Objetos temporários**
  - Podem ser criados com o construtor

```
int main()
{
    // criação de objeto
    Jogo mk;

    // usando construtor
    // para criar um
    // temporário
    mk = Jogo("MK", 80.0f);
}
```

Objeto  
Temporário





# Temporários

## ■ Objetos temporários

Temporário	
Criação	Destruição
chamada do construtor	após a atribuição

Jogo	Memória	
	nome	"MK" 0xCB2B = mk
	preco	80.0 0xCB2F
	horas	0 0xCB33
	custo	0 0xCB37

```
int main()
{
    // criação de objeto
    Jogo mk;

    // usando construtor
    // para criar um
    // temporário
    mk = Jogo("MK", 80.0f);
}
```

Objeto  
Temporário

Construtor  
Destrutor

# Resumo

- **Destrutores** permitem a **liberação de recursos**
  - São chamados no **final da vida** do objeto
  - O compilador cria um destrutor padrão
    - Se um não for definido

```
// destrutor padrão  
Jogo::~~Jogo()  
{  
    // vazio  
}
```



Não precisa ser  
fornecido se não houver  
alocação de recursos

# Resumo

- Um **destrutor** **deve ser fornecido** para:

- Liberar memória alocada
- Fechar arquivos abertos
- Encerrar conexões estabelecidas
- Etc.

Essencial para a  
liberação de **memória**  
**alocada** com **new**

```
Conjunto::Conjunto(int n)
{
    // aloca memória
    vet = new int[n];
}
```

```
Conjunto::~~Conjunto()
{
    // libera memória
    delete [] vet;
}
```

