

Programação Orientada a Objetos

CONSTRUTORES

✓ *em*

C++

Introdução

- Um dos objetivos do C++ é fazer os **objetos** funcionarem como tipos primitivos, porém:

```
✓ int ano = 2022;  
  
struct Tempo  
{  
    int horas;  
    int minutos;  
    int segundos;  
};  
  
✓ Tempo t = {2, 30, 45};
```

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

✗ Jogo gears = { "Gears", 50.0f, 200, 0.25f };

Inicialização inválida:
membros
inacessíveis

Introdução

- Temos contornado isso com **métodos de criação**
 - Inicializam os objetos

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

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

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

Jogo gears;

gears.adquirir("Gears", 50.0f);

Introdução

- Assumimos que os métodos de criação são chamados
 - Mas não existe uma forma de impor isso

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

public:
    void adquirir(const string & nome, float valor);
    void atualizar(float valor);
    void exibir();
    void jogar(int tempo);
};
```



```
Jogo gears;
gears.adquirir("Gears", 50.0f);
gears.jogar(2);
gears.exibir();
```

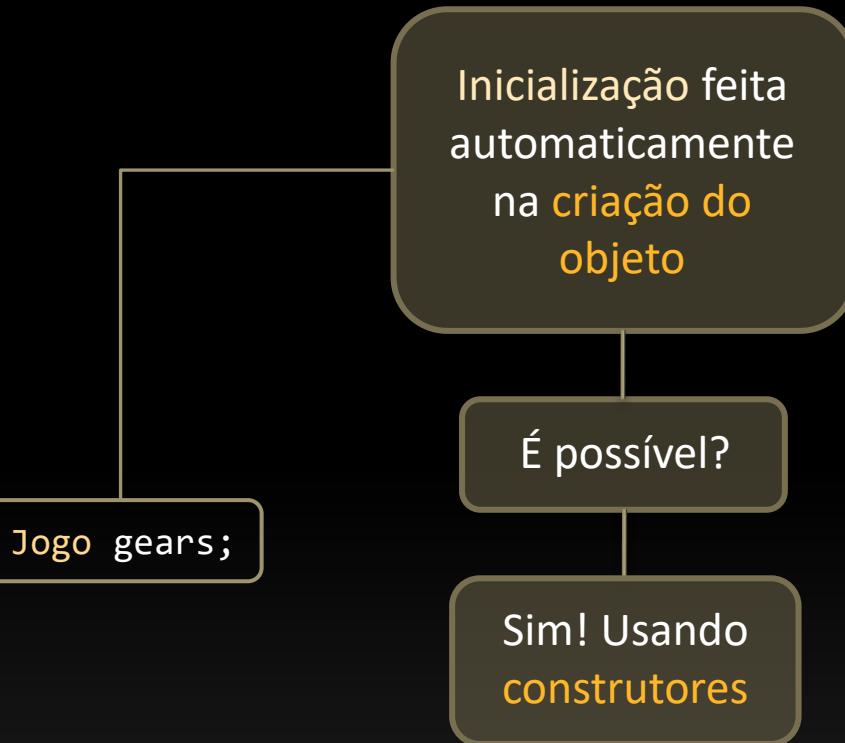
Introdução

- A solução é **inicializar os objetos**:

- Automaticamente
- Na sua criação

JOGO

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



Jogo gears;

Construtores

- C++ fornece uma **função membro especial**
 - Chamada de **construtor**
 - Possui o mesmo nome da classe
 - Não possui retorno

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

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

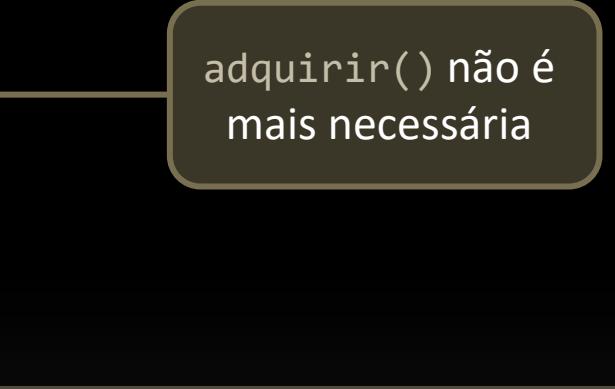
Jogo gears;

Construtores

- O protótipo deve ser adicionado a classe

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

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



adquirir() não é mais necessária

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



Então como vamos fornecer um nome para o jogo?

Construtores

- **Construtores** podem ter **parâmetros**
 - Inclusive com **argumentos 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);
};
```

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

Parâmetros do Construtor

- **Cuidado com os nomes dos parâmetros**
 - Queremos que eles sejam significativos
 - Mas eles não podem ser iguais aos atributos

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

    ...
};
```

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

Não modificam
atributos

Parâmetros do Construtor

- Existem algumas convenções comuns
 - Tipicamente preservam os nomes dos parâmetros
 - Modificam membros para usar prefixos ou sufixos

```
class Jogo
{
private:
    string m_nome;
    float m_preco;
    int m_horas;
    float m_custo;
    ...
};
```

```
class Jogo
{
private:
    string nome_;
    float preco_;
    int horas_;
    float custo_;
    ...
};
```

```
Jogo::Jogo(const string & nome,
            float preco)
{
    nome_ = nome;
    preco_ = preco;
    horas_ = 0;
    custo_ = preco;
}
```

Parâmetros do Construtor

- Existem **outras alternativas**
 - Modificam os nomes dos parâmetros
 - Utilizam nomes correlatos, sinônimos, etc.

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

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



Nomes
Diferentes
dos Atributos

Usando Construtores

- C++ fornece várias formas de inicializar um objeto
 - Chamando o construtor

```
// explicitamente  
Jogo gears = Jogo("Gears", 50.0f);  
  
// implicitamente  
Jogo gears("Gears", 50.0f); ⚡
```

- Usando inicialização por listas

```
// disponível a partir do C++11  
Jogo gears { "Gears", 50.0f };  
Jogo gears = { "Gears", 50.0f };
```

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

Usando Construtores

- Se o **construtor** não possuir parâmetros
 - Ou todos possuírem argumentos padrão

```
class Point
{
private:
    int x;
    int y;

public:
    Point() { x = 0; y = 0; }
    void MoveTo(int px, int py);
    void Translate(int dx, int dy);
};
```

```
// opções válidas
✓ Point p;
✓ Point p {};
✓ Point p = {};
✓ Point p = Point();

// ambiguidade para o compilador:
// objeto p ou protótipo de função
// que retorna um Point?
✗ Point p();
```

Usando Construtores

- **Construtores não são métodos da classe**
 - Não é possível usar um objeto para invocar o construtor
 - Ele é chamado automaticamente

```
class Point
{
private:
    int x;
    int y;

public:
    Point() { x = 0; y = 0; }
    void MoveTo(int px, int py);
    void Translate(int dx, int dy);
};
```

```
// cria objeto
Point p;

// chama métodos
✓ p.MoveTo(10,50);
✓ p.Translate(2,8);

// construtor não pode
✗ p.Point();
```

Usando Construtores

- Toda **criação de objeto** vai chamar o construtor
 - Se a classe possuir um construtor

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

public:
    void adquirir(const string & nome, float valor);
    void atualizar(float valor);
    void exibir();
    void jogar(int tempo);
};
```



E se a classe
não possuir um
construtor?

```
Jogo gears;
gears.adquirir("Gears", 50.0f);
```

Construtores Padrão

- Toda classe possui um **construtor padrão**
 - Um construtor que não recebe valores

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

public:
    Jogo();
    void adquirir(const string & titulo, float valor);
    void atualizar(float valor);
    ...
};
```

Se sua classe não possuir, o compilador cria um para você

```
// construtor padrão
Jogo::Jogo()
{
    // vazio
}
```

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

Construtores Padrão

- O **compilador** fornece um **construtor padrão** apenas se não existir nenhum construtor

```
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);
};
```

```
// criação do objeto
✓ Jogo gears = Jogo("Gears");
✓ Jogo doom { "Doom" };

// não existe construtor padrão
✗ Jogo gta = Jogo();
✗ Jogo rdr2;
```

Mas por que não criar um automaticamente?

Construtores Padrão

- A ausência do construtor padrão
 - Impede a criação de objetos não inicializados

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

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



Memória	
nome	\$#!%@*
preco	8730.2
horas	7542.0
custo	-98.2803

// não existe construtor padrão
✗ Jogo gears;

Construtores Padrão

- É possível fornecer **mais de um construtor**
 - Usando **sobrecarga de funções**

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

public:
    Jogo();
    Jogo(const string & titulo, float valor = 0);
    ...
};
```

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

// criação de objetos
✓ Jogo gears = Jogo("Gears", 50.0f);
✓ Jogo doom { "Doom" };
✓ Jogo gta = Jogo();
✓ Jogo rdr2;
```

Resumo

- **Construtores** permitem a **inicialização** de objetos
 - São chamados na criação do objeto
 - O compilador cria um vazio se não existir nenhum

```
// criação do objeto
Jogo gears;

// construtor padrão
Jogo::Jogo()
{
    // vazio
}
```



A diagram illustrating memory allocation. A pointer variable 'gears' of type 'Jogo*' points to a memory location at address 0xCB2B. This location contains a 'Jogo' object with the following fields:

Memória	
	JOGO
nome	\$#!%@*
preco	8373922
horas	7464269
custo	-98.2803



Lixo na Memória

Resumo

- É recomendado fornecer um construtor padrão
 - Que inicializa todos os atributos
 - A não ser que não exista bons valores padrão para os objetos
 - Crie construtores com parâmetros adequados
 - Não forneça um construtor padrão

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

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