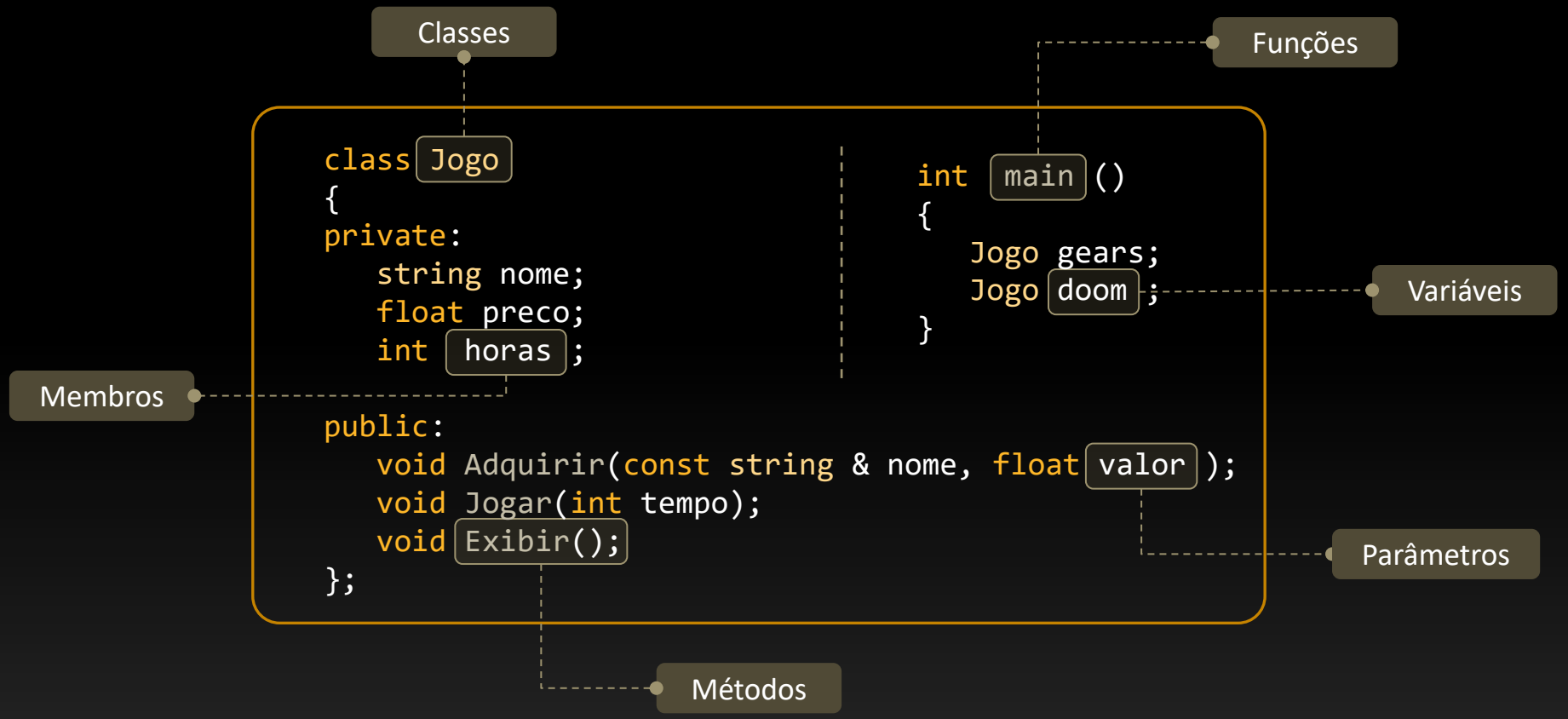
Programação Orientada a Objetos

NAMESPACES

em C++

Introdução

- Nomes em C++ podem ser dados a:



Introdução

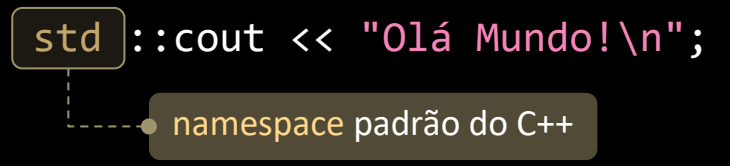
- Conflitos de nomes são comuns quando os projetos:
  - Crescem em tamanho e complexidade
  - Utilizam várias bibliotecas

```
#include <cmath>          // Standard C++ Math Library
#include <mkl>            // Intel Math Kernel Library
#include <gls>            // GNU Scientific Library
#include <DirectXMath>    // DirectX Math Library
```

Imagine utilizar várias bibliotecas matemáticas, sendo que todas fornecem uma constante pi e funções de mesmo nome sin( ) e cos( )...

Introdução

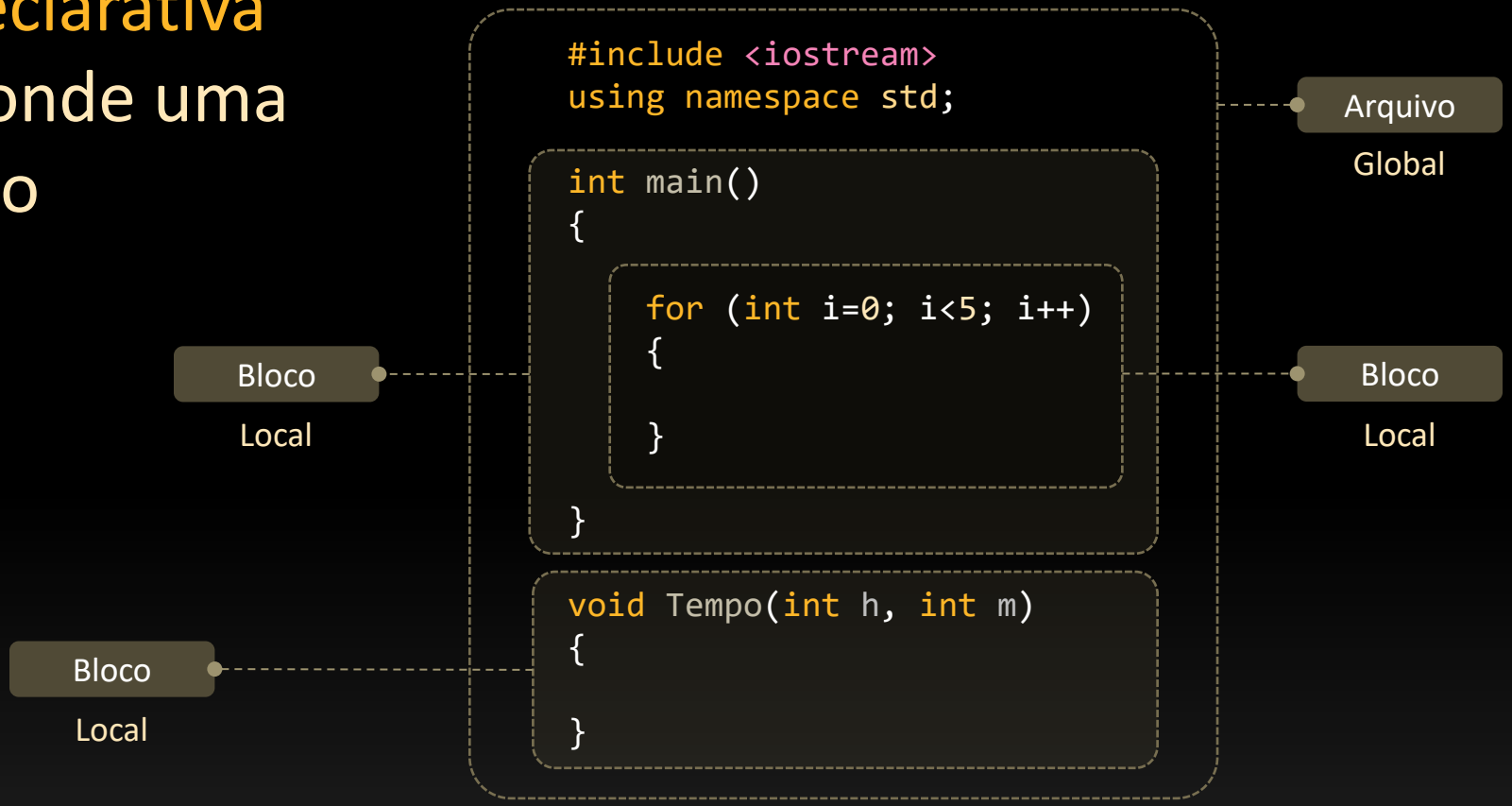
- C++ fornece namespaces para o programador ter maior controle sobre o escopo de nomes



  - Para entendermos melhor precisamos definir:
    - Região declarativa
    - Escopo potencial
    - Escopo

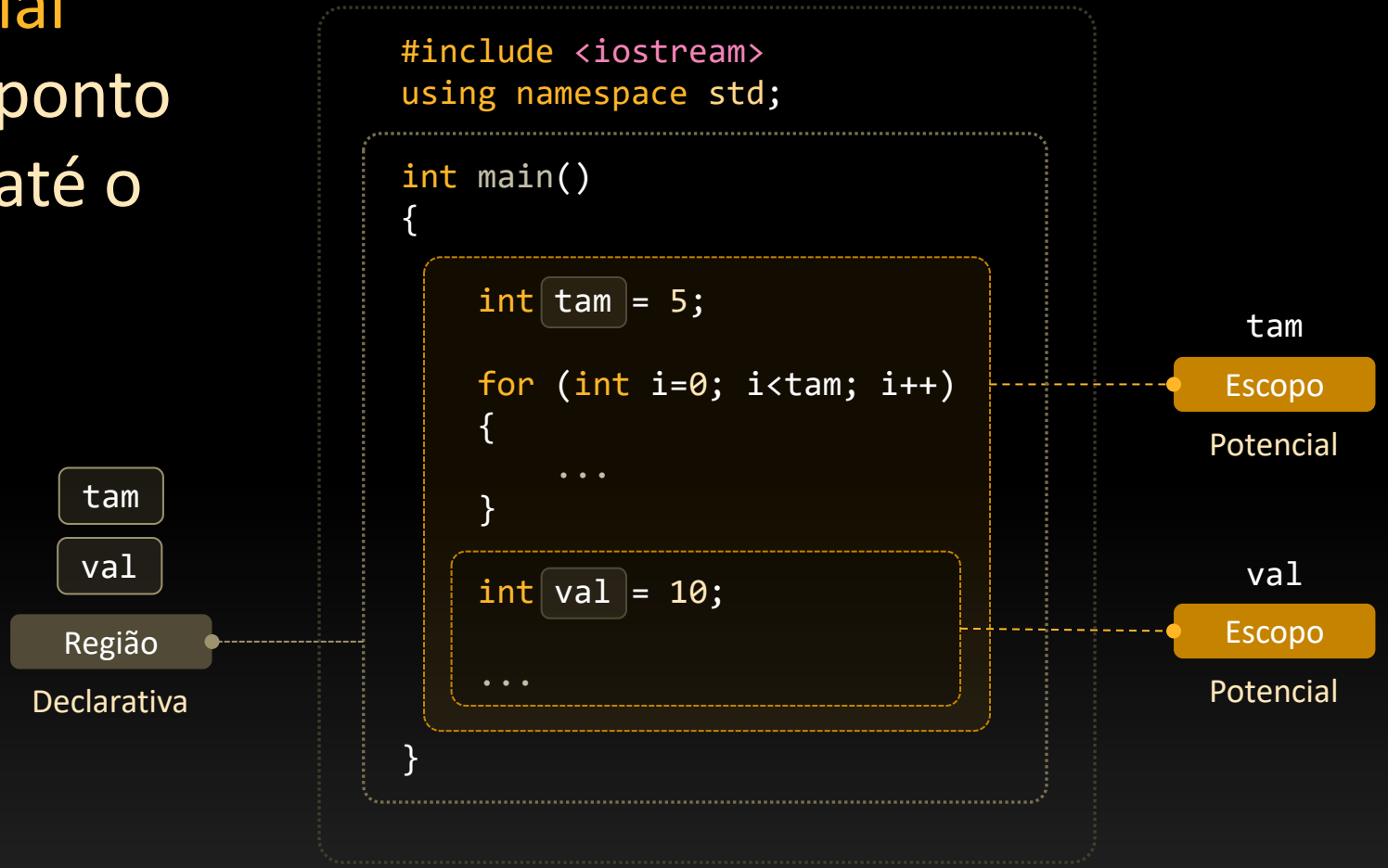
Conceitos

- Região declarativa é o local onde uma declaração pode ser feita



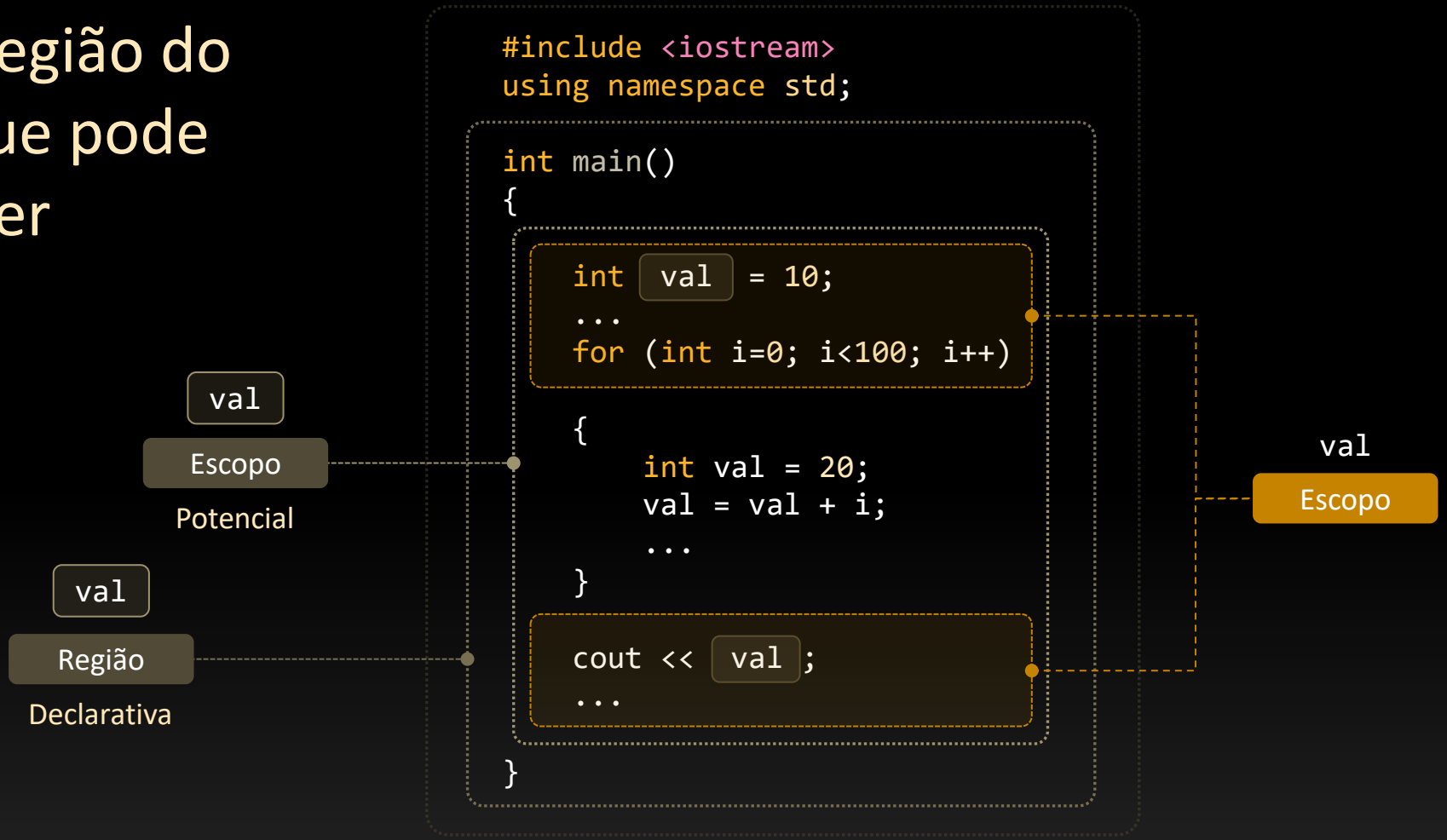
Conceitos

- Escopo potencial se estende do ponto de declaração até o final da região declarativa



Conceitos

- Escopo é a região do programa que pode realmente ver o nome



Namespace

- Um namespace é uma nova região declarativa
  - Nomes declarados em um namespace:
    - Não conflitam com nomes em outros namespaces
    - São acessados através do operador de resolução de escopo ::

```
namespace Volt
{
    int fps;
    float time;
}
```

```
int main()
{
    Volt::fps = 60;
    Volt::time = 0.016f;
}
```

Namespace

- Nomes declarados em um namespace:
  - Possuem ligação externa por padrão
    - Não podem ficar dentro de blocos

```
int main()
{
    namespace Volt
    {
        int fps;
        float time;
        const double Pi = 3.14;
    }

}
```

Constantes sempre possuem ligação interna, são visíveis apenas no arquivo em que foram declaradas.

Namespace

- Nomes fora de um namespace:
  - Estão no namespace global
    - Acessado através de ::

```
// variável global
int count;

namespace Volt
{
    float time;
    int fps;
}
```

```
int main()
{
    ::count = 0;   // usando namespace global
    count = 1;     // :: é opcional

    Volt::time = 0.016f;
    Volt::fps = 60;
}
```

Namespace

- É possível adicionar nomes a um namespace existente

Volt.h

```
namespace Volt
{
    // variáveis
    float time;
    int fps;

    // protótipo da função
    void FrameTime();
}
```

Volt.cpp

```
namespace Volt
{
    // adicionando variável
    int count;

    // adicionando definição da função
    void FrameTime()
    { ... }
}
```

Acessando Nomes

- Para acessar nomes em um namespace:
  - Precisamos "qualificar" o nome

```
int main()
{
    Volt::time = 0.016f;
    Volt::fps = 60;
    Volt::count = 0;
    Volt::FrameTime();
}
```

  - Sem `Volt::` o nome é dito "não-qualificado"

Acessando Nomes

- Usar o nome qualificado nem sempre é conveniente
  - Para simplificar, C++ fornece:
    - Declarações `using`
      Tornam alguns nomes disponíveis

```
using Volt::time;
using Volt::FrameTime;
```

    - Diretivas `using`
      Tornam todos os nomes disponíveis

```
using namespace Volt;
```

Declarações using

- Uma declaração `using` adiciona um nome na região declarativa em que ela se encontra

```
namespace Volt
{
    float time;
    int count;
    int fps;
    void FrameTime();
}

char fps;  // namespace global
```

```
int main()
{
    using Volt::fps;

    cin >> fps;     // int
    cin >> ::fps;   // char

    // erro, fps já existe!
    double fps;  ❌

}
```

Região Declarativa

Declarações using

- As regras de visibilidade de nomes continuam valendo
  - Nomes locais escondem os globais

```
namespace Volt
{
    float time;
    int count;
    int fps;
    void FrameTime();
}

char fps;  // namespace global
```

```
int main()
{
    // globais
    cin >> Volt::fps;   // int
    cin >> ::fps;       // char

    // local
    double fps;         // ok
    cin >> fps;         // double
}
```

Declarações using

- Uma declaração using fora de funções
  - Adiciona nomes ao namespace global

```
namespace Volt
{
    struct Stamp
    {
       ...
    };

    float time;
    int count;
    int fps;
    void FrameTime();
}
```

```
using Volt::fps;     // região arquivo

int main()
{
    cin >> fps;      // Volt::fps
    Show();
}

int Show()
{
    cout << fps;      // Volt::fps
}
```

Diretiva using

- A diretiva using põe todos os nomes na região declarativa
  - Se diferencia pelo uso da palavra namespace

```
namespace Volt
{
    struct Stamp
    {
        ...
    };

    float time;
    int count;
    int fps;
    void FrameTime();
}
```

```
int main()
{

    using namespace Volt;

    Stamp s { 0, 0 };
    time = 0.016f;
    count = 0;
    fps = 60;
    FrameTime();
}
```

Região Declarativa

Diretiva using

- Uma diretiva using fora de funções
  - Torna todos os nomes acessíveis globalmente

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

int main() {
    int num = 0;
    cin >> num;
    Exibir(num);
}

void Exibir(int n) {
    cout << "Olá Mundo!\n";
}
```

Temos feito isso para acessar cin e cout de forma fácil por todo nosso código.

Diretiva using

- A diretiva deve ser usada com cuidado
  - Aumenta a chance de conflitos
  - Perigosa em códigos:
    - Grandes e complexos
    - Usam muitas bibliotecas

```
using std::cout;
using std::cin;
using std::endl;
```

```
#include <iostream>

using namespace std;

int main() {
    int num = 0;
    cin >> num;
    Exibir(num);
}

void Exibir(int n) {
    cout << "Olá Mundo!\n";
}
```

Diretiva versus Declaração

- Existem mais diferenças importantes
  - Declaração using: é como se o nome tivesse sido declarado na posição do using

```
int main()
{
    using Volt::fps;

    cin >> fps;

    // fps já existe!
    double fps;  ❌
}
```

Impede que uma variável local de mesmo nome seja criada, como se `fps` fosse uma variável local.

Diretiva versus Declaração

- Existem mais diferenças importantes
  - Diretiva using: é como se o nome tivesse sido declarado na posição em que o namespace foi declarado

```
int main()
{
    using namespace Volt;

    cin >> fps;

    // esconde a global
    double fps;
}
```

Traz `fps` com o comportamento normal de uma variável global, permitindo variáveis locais com mesmo nome.

Diretiva versus Declaração

- A declaração using é mais segura
  - Mostra exatamente os nomes incluídos

```
using Volt::time;
using Volt::fps;
using Volt::FrameTime;
```

  - Se o nome conflita, o compilador avisa

```
int main()
{
    using Volt::fps;
    double fps; ❌
}
```

Diretiva versus Declaração

- A diretiva using é mais simples
  - Deixa o código mais limpo
  - Conveniente para programas pequenos

```
using namespace Volt;

int main()
{
    Stamp s { 0, 0 };
    time = 0.016f;
    count = 0;
    fps = 60;
    FrameTime();
}
```

Dispensa o uso do operador :: nos acessos aos membros do namespace.

Namespaces Aninhados

- O aninhamento permite categorizar nomes

```
namespace Volt
{
    namespace Window {
        int color;
        int width;
        int height;
    }

    namespace Graphics {
        float time;
        int fps;
    }
}
```

```
using namespace Volt::Window;
using Volt::Graphics::time;

int main()
{
    color = 250;
    width = 1920;
    height = 1080;

    time = 0.016f;

    Volt::Graphics::fps = 60;
}
```

Namespaces Aninhados

- Declarações e diretivas using podem ser usadas dentro de namespaces

```
namespace Window
{
    int color;
    int width;
    int height;
}

namespace Graphics
{
    float time;
    int fps;
}
```

```
namespace Volt {

    using Window::color;
    using Window::width;
    using Window::height;

    using namespace Graphics;
}
```

Agrupa namespaces mas sem torná-los em subcategorias.

Namespaces Aninhados

- Declarações e diretivas using podem ser usadas dentro de namespaces

```
namespace Volt {

    using Window::color;
    using Window::width;
    using Window::height;

    using namespace Graphics;
}
```

Agrupa namespaces mas sem torná-los em subcategorias.

```
using namespace Volt;

int main()
{
    color = 250;
    width = 1920;
    height = 1080;

    time = 0.016f;
    fps = 60;
}
```

Namespaces Aninhados

- Apelidos podem ser criados
  - Simplificam aninhamentos

```
namespace Volt
{
    namespace Window {
        int color;
        int width;
        int height;
    }

    namespace Graphics {
        float time;
        int fps;
    }
}
```

```
namespace Engine = Volt;

namespace Win = Engine::Window;

int main()
{
    Win::color = 250;
    Win::width = 1920;
    Win::height = 1080;

    Engine::Graphics::time = 0.016f;
    Engine::Graphics::fps = 60;
}
```

Resumo

- Namespaces simplificam:
  - Gerenciamento de nomes em grandes projetos
    - Evitam choques
    - Destacam de onde vem

```
Engine::Graphics::fps = 60;
```

  - O uso de nomes em pequenos projetos
    - Não requer digitação extra
    - Torna o código mais limpo

```
using namespace std;
```