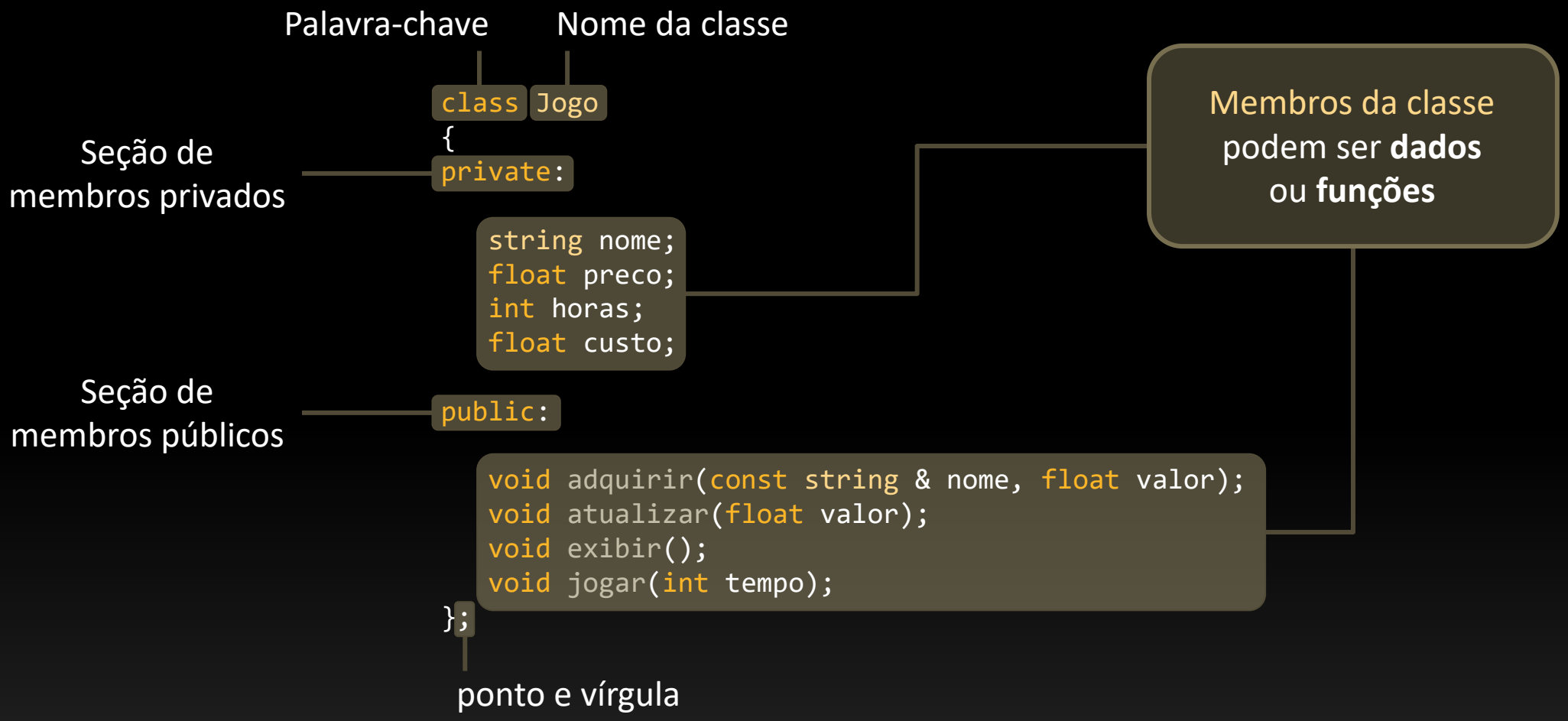
Programação Orientada a Objetos

FUNÇÕES MEMBRO

em C++

Introdução


Palavra-chave
Nome da classe
class Jogo
{
Seção de membros privados
private:
string nome;
float preco;
int horas;
float custo;
Membros da classe podem ser **dados** ou **funções**
Seção de membros públicos
public:
void adquirir(const string & nome, float valor);
void atualizar(float valor);
void exibir();
void jogar(int tempo);
};
ponto e vírgula

Introdução

- As palavras-chave `public` e `private` fornecem controle de acesso aos membros da classe
  - Public: membros públicos podem ser acessados **diretamente**
  - Private: membros privados só podem ser acessados **através dos métodos**

```
class Jogo
{
public:
  int horas;
  ...
```

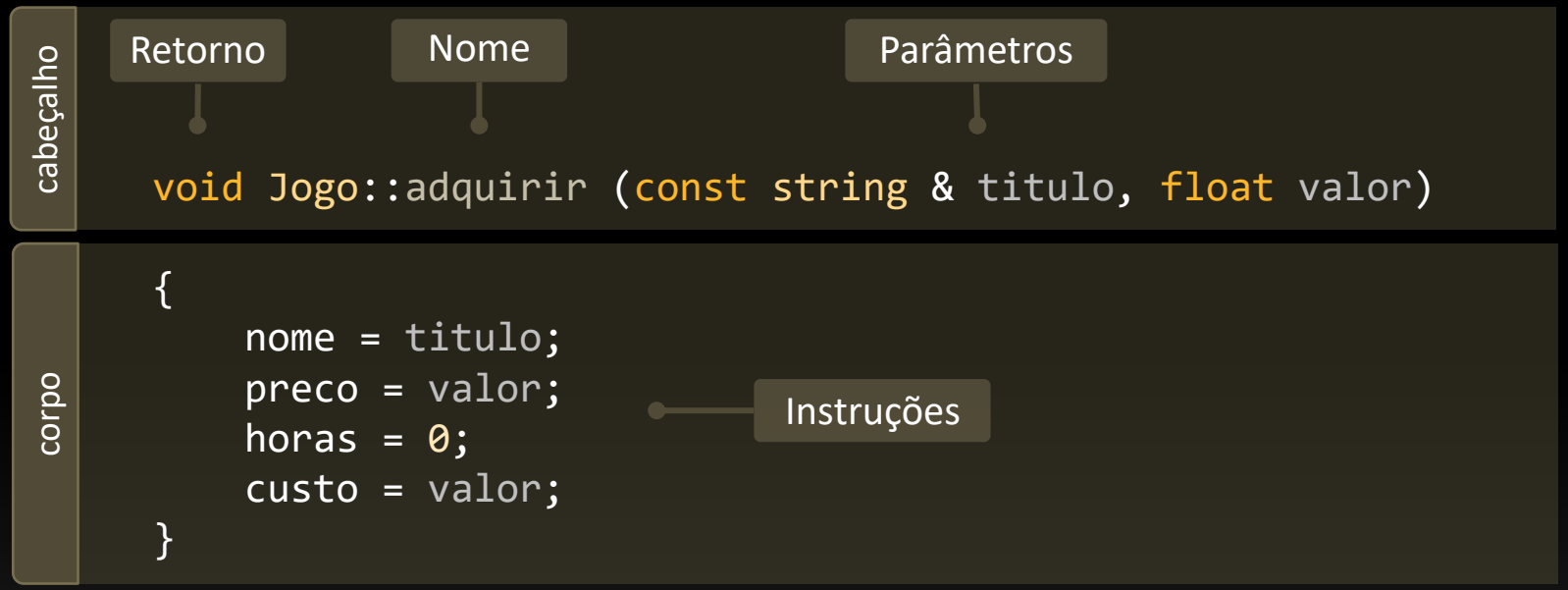
```
Jogo gow;
✓ gow.horas = 5;
✓ gow.jogar(5);
```

```
class Jogo
{
private:
   int horas;
   ...
```

```
Jogo gow;
✗ gow.horas = 5;
✓ gow.jogar(5);
```

Funções Membro

- **Métodos** ou funções membro são funções normais



Funções Membro

- Se diferem apenas por duas características especiais
  - Acesso aos membros privados da classe
  - Uso do operador de resolução de escopo ::

Operador de Resolução de Escopo

```
void Jogo :: exibir()
{
    cout << nome  << " R$"
         << preco << " "
         << horas << "h = R$"
         << custo << "/h\n";
}
```

```
class Jogo
{
private:

   string nome;
   float preco;
   int horas;
   float custo;

public:

   void exibir();
   ...
};
```

Operador ::

- O operador de resolução de escopo
  - Indica a que classe o método pertence

Jogo.h

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

public:
   void exibir();
   ...
};
```

Jogo.cpp

```
#include "Jogo.h"

void Jogo::exibir()
{
    cout << nome  << " R$"
         << preco << " "
         << horas << "h = R$"
         << custo << "/h\n";
}

...
```

Os métodos são normalmente definidos fora da classe

Operador ::

- O operador de resolução de escopo
  - Permite ter o mesmo nome de função em classes diferentes


```
void Jogo ::exibir()
{
    cout << nome  << " R$"
         << preco << " "
         << horas << "h = R$"
         << custo << "/h\n";
}
```

```
void Coord ::exibir()
{
    cout << "(" << x
         << "," << y
         << ")";
}
```

Operador ::

- O operador :: não precisa ser usado:
  - Dentro da declaração da classe

```
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);
};
```

Todos os membros possuem escopo de classe.

Operador ::

- O operador :: não precisa ser usado:
  - Para acessar membros dentro de funções membro

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

   void calcular()
   {
      if (horas > 0)
         custo = preco/horas;
   }
   ...
```

```
void Jogo::atualizar(float valor)
{
    preco = valor;

    calcular();
}

void Jogo::jogar(int tempo)
{
    horas = horas + tempo;
    calcular();
}
```

Operador ::

- O nome completo de um método
  - Inclui o nome da classe
  - É chamado de nome qualificado

```
void Jogo::jogar (int tempo)
{
    horas = horas + tempo;
    calcular();
}
```

Nomes não qualificados, ou abreviados, podem ser usados apenas no escopo da classe.

Métodos Inline

- **Métodos** definidos na declaração da classe se tornam automaticamente inline

Tipicamente implementam funções curtas que são chamadas com frequência.

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

   void calcular()
   {
      if (horas > 0)
         custo = preco/horas;
   }

   ...
};
```

Métodos Inline

- Eles podem ser definidos também fora da classe
  - Usando a palavra-chave inline

Jogo.h

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

   void calcular();

   ...
};
```

Jogo.cpp

```
#include "Jogo.h"

inline void Jogo::calcular()
{
   if (horas > 0)
      custo = preco/horas;
}

...
```

Métodos Inline

- As regras da linguagem obrigam métodos inline a serem definidos em cada arquivo .cpp que os utiliza

Jogo.h

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

   void calcular();

   ...
};
```

JogoX.cpp

```
#include "Jogo.h"

inline void Jogo::calcular()
{
   if (horas > 0)
      custo = preco/horas;
}

...

// chamada a calcular()
```

JogoY.cpp

```
#include "Jogo.h"

inline void Jogo::calcular()
{
   if (horas > 0)
      custo = preco/horas;
}

...

// chamada a calcular()
```

Métodos Inline

- A forma mais fácil de cumprir a regra para um projeto com vários arquivos:
  - Definir os métodos inline no arquivo .h que contém a declaração da classe
  - Perde-se a separação da interface e implementação da classe

```
// Jogo.h //

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

   void calcular();
   ...
};
```

```
inline void Jogo::calcular()
{
   if (horas > 0)
      custo = preco/horas;
}
```

Objetos e Métodos

- **Métodos** acessam dados privados de um objeto
  - Mas de qual objeto?

```
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);
};
```

```
void Jogo::exibir()
{
    cout << nome  << " R$"
         << preco << " "
         << horas << "h = R$"
         << custo << "/h\n";
}
```

Estes dados pertencem a qual objeto?

Objetos e Métodos

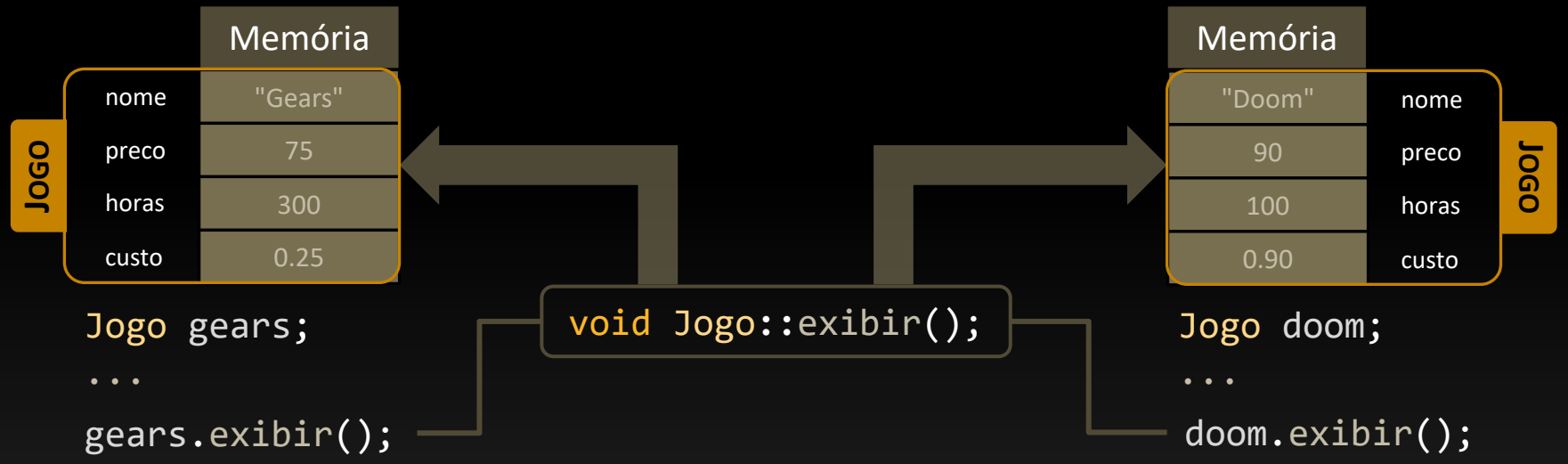
- Todos os **objetos** vão ter sua cópia dos atributos
  - Os métodos são chamados a partir dos objetos
    - Usando o operador membro (.)

```
int main()
{
    Jogo gears;
    Jogo doom;
    ...
    gears.exibir();
    doom.exibir();
}
```

```
void Jogo::exibir()
{
    cout << nome  << " R$"
         << preco << " "
         << horas << "h = R$"
         << custo << "/h\n";
}
```

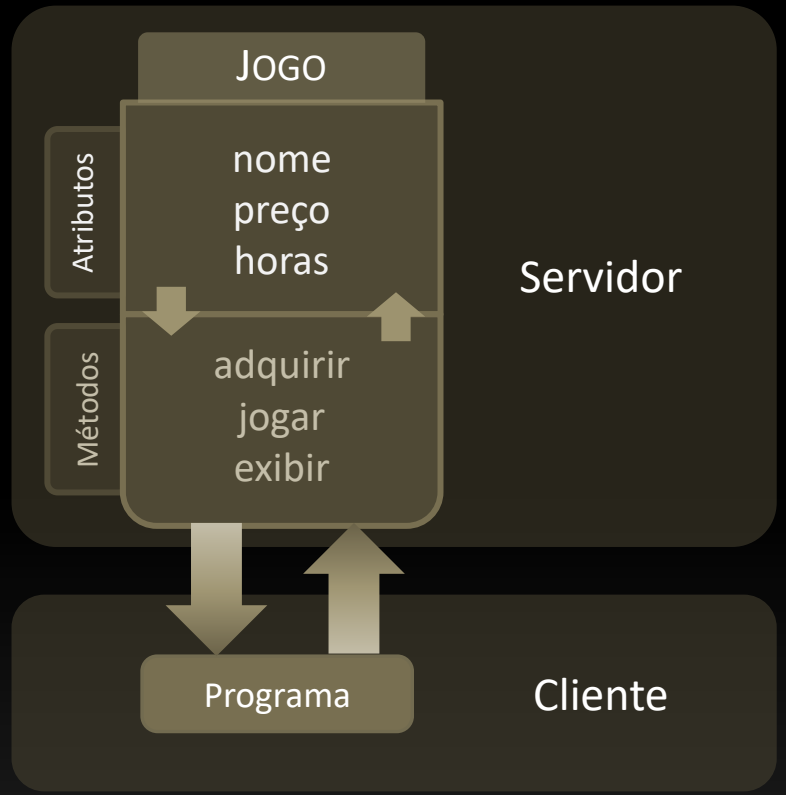
Objetos e Métodos

- **Objetos** de uma classe:
  - Tem sua própria cópia dos atributos
  - Compartilham o mesmo conjunto de métodos



Orientação a Objetos em C++

- Modelo cliente/servidor
  - Cliente é o usuário da classe
    - Deve conhecer a interface (métodos públicos)
  - Servidor é o criador da classe
    - Deve implementar as operações de acordo com a interface
    - Mudanças devem ser feitas nos detalhes de implementação (não na interface)



Orientação a Objetos em C++

- **Objetos** podem:
  - Ser passados como argumentos
  - Retornados em funções
  - Atribuídos para outros objetos

- A **linguagem** permite:
  - Inicializar objetos
  - Ensinar cin e cout a reconhecer objetos
  - Fazer conversões automáticas entre objetos

O objetivo da linguagem é fazer o uso de objetos e classes similar ao uso de variáveis e tipos primitivos.

Resumo

- **Métodos** são funções normais
  - O operador de escopo ::
    - Indica a qual classe um método pertence
    - Não precisa ser usado para membros no escopo da classe
    - A implementação dos métodos está no escopo da classe

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

```
void Jogo::exibir()
{
    cout << nome  << " R$"
         << preco << " "
         << horas << "h = R$"
         << custo << "/h\n";
}
```

Resumo

- **Métodos Inline** podem ser definidos
  - Na declaração da classe
  - Nos arquivos .cpp
  - No arquivo .h

```
inline void Jogo::calcular()
{
   if (horas > 0)
      custo = preco/horas;
}
```

```
// Jogo.h

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

   void calcular();
   ...
};
```

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

   void calcular()
   {
      if (horas > 0)
         custo = preco/horas;
   }
   ...
};
```