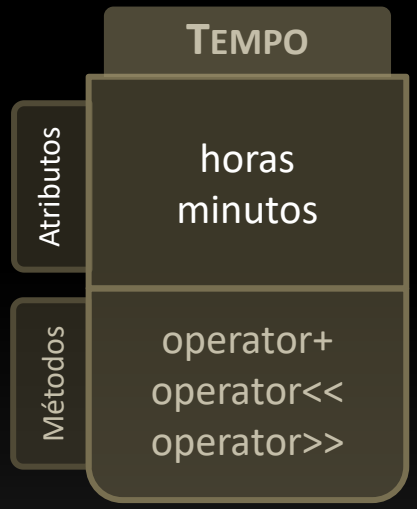
Programação Orientada a Objetos

CONVERSÕES DE TIPO PARA CLASSES

em C++

Introdução

- Uma classe define um **novo tipo**
  - Funciona como um tipo primitivo




```
int main()
{
    Tempo t1 = {2, 10};
    Tempo t2 = {1, 30};

    cout << t1 + t2;

}
```

Sobrecarga de Operadores
+
Funções Amigas

Introdução

- Mas tipos primitivos podem ser convertidos
  - Existem **regras** para fazer conversões
    - Automáticas

```
long cont = 8;          ✓   // int convertido para long
float total = 11;       ✓   // int convertido para float
int lado = 3.5;         ✓   // double convertido para int
int * p = 10;           ✗   // não há conversão automática
```

    - Manuais

```
int * p = (int *) 10;   ✓   // int convertido para endereço
```

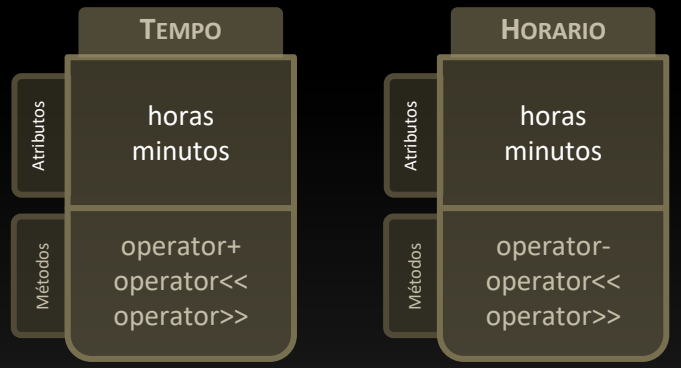
Conversões de Tipos

- Se a **classe** é um tipo primitivo...
  - Ela deve permitir conversões
  - Mas quando elas fazem sentido?

1. Duas classes bastante relacionadas

```
Horario inicio { 18, 30 };
Horario fim { 21, 00 };

// Horario convertido para Tempo
Tempo duracao = fim - inicio;
```



Conversões de Tipos

- Se a **classe** é um tipo primitivo...
  - Ela deve permitir conversões
  - Mas quando elas fazem sentido?

> 2. Classe relacionada a um **tipo primitivo**

```
Tempo ida = 2;                  // int convertido para Tempo
Tempo volta = 1.5;              // double convertido para Tempo
float total = ida + volta;      // Tempo convertido para float
```

Conversões com Construtores

- **Conversões** são feitas com construtores
  - Que recebem apenas um argumento

```
class Tempo
{
private:
   int horas;
   int minutos;

public:
   Tempo();
   Tempo(int h);
   Tempo(int h, int m);
   ...
};
```

```
int main()
{
   // cria objeto
   Tempo t;

   // converte int para Tempo

   t = 3 ;

}
```

Chama Construtor

`Tempo(3)`

Conversões com Construtores

- A **conversão** pode ser implícita ou explícita
  - O construtor é chamado em ambos os casos

```
int main()
{
   // cria objetos
   Tempo t1, t2, t3;

   // converte int para Tempo
   t1 = Tempo(3);
   t2 = (Tempo) 3;
   t3 = 3;
}
```

Conversões com Construtores

- Verificando as chamadas do construtor

```
int main()
{
   Tempo t1 = Tempo(1, 10);
   Tempo t2 = Tempo(2);
   Tempo t3;

   // converte int para Tempo
   t3 = 3;
}
```

Saída:

```
Construtor Hora-Min
Construtor Hora
Construtor Padrão
Construtor Hora
```

```
Tempo::Tempo() {
    horas = minutos = 0;
    cout << "Construtor Padrão\n";
}

Tempo::Tempo(int h) {
    horas = h; minutos = 0;
    cout << "Construtor Hora\n";
}

Tempo::Tempo(int h, int m) {
    horas = h; minutos = m;
    cout << "Construtor Hora-Min\n";
}
```

Conversões com Construtores

- Usando a inicialização por { } do C++11
  - Conversões funcionam com múltiplos argumentos

```
class Tempo
{
private:
   int horas;
   int minutos;

public:
   Tempo();
   Tempo(int h);
   Tempo(int h, int m);
   ...
};
```

```
int main()
{
   // cria objetos
   Tempo t0, t1, t2, t3;

   t3 = 3;           // int para Tempo
   t2 = { 1, 10 };   // {int,int} para Tempo
   t1 = { 5 };       // {int} para Tempo
   t0 = {};          // {} para Tempo
}
```

Conversões com Construtores

- Argumentos padrão podem ser usados
  - A conversão acontece para todas as combinações válidas

```
class Tempo
{
private:
   int horas;
   int minutos;

public:
   Tempo(int h = 0, int m = 0);
   Tempo(int h);
   Tempo(int h, int m);
...
};
```

```
int main()
{
   // cria objetos
   Tempo t0, t1, t2, t3;

   t3 = 3;           // int para Tempo
   t2 = { 1, 10 };   // {int,int} para Tempo
   t1 = { 5 };       // {int} para Tempo
   t0 = {};          // {} para Tempo
}
```

Conversões Implícitas

- As conversões automáticas parecem legais
  - Mas nem sempre são desejadas
  - É possível desligar conversões implícitas

```
class Tempo
{
private:
   int horas;
   int minutos;

public:

   explicit Tempo(int h = 0,
                  int m = 0);

   ...
};
```

```
int main()
{
   Tempo t1, t2, t3;

   t1 = Tempo(3);     ✓   // explícita
   t2 = (Tempo) 3;    ✓   // explícita
   t3 = 3;            ✗   // implícita
}
```

Conversões Implícitas

- Onde são feitas conversões implícitas?
  - Inicialização de objetos

```
Tempo t0 = {};          // {} para Tempo
Tempo t1 = { 5 };       // {int} para Tempo
Tempo t2 = { 1, 10 };   // {int,int} para Tempo
Tempo t3 = 3;           // int para Tempo
```

Objetos já são criados com os valores da inicialização.

  - Atribuição para objetos

```
Tempo t0, t1, t2, t3;
t0 = {};                // {} para Tempo
t1 = { 5 };             // {int} para Tempo
t2 = { 1, 10 };         // {int,int} para Tempo
t3 = 3;                 // int para Tempo
```

Cria objetos **temporários** e depois realiza cópia dos atributos.

Conversões Implícitas

- Onde são feitas conversões implícitas?
  - Chamada de função

```
// protótipo da função
void Exibir(Tempo t);
...
Exibir(3);
```

  - Retorno de função

```
Tempo SomarHoras(Tempo a, Tempo b) {
    // a soma é um inteiro
    return a.horas + b.horas;
}
```

Objetos temporários são criados em ambos os casos.

Conversões Implícitas

- Onde são feitas conversões implícitas?
  - Em qualquer das situações anteriores
    - Se um tipo puder ser convertido, sem ambiguidade, para o tipo do parâmetro do construtor

```
// inicialização
Tempo t1 = 1.5f;

// atribuição
Tempo t2;
t2 = 2L;

// chamada de função
void Exibir(Tempo t);
Exibir(3.1);
```

```
// retorno de função
Tempo SomarHoras(Tempo a,
                 Tempo b)
{
    short resultado =
        a.horas +
        b.horas;

    return resultado;
}
```

Funções de Conversão

- **Convertemos** inteiros para Tempo
  - É possível fazer o contrário?

```
Tempo ida { 2, 10 };
Tempo volta { 2, 30 };

int horas = ida;         // 2 horas
double valor = volta;    // 2.5 horas
```

É possível através de uma função de conversão

Funções de Conversão

- As **funções de conversão** agem como um type cast
  - Uma conversão de Tempo para double permite:

```
Tempo trecho1 { 1, 10 };
Tempo trecho2 { 2, 50 };
Tempo viagem;

viagem = trecho1 + trecho2;

// conversão explícita
double parcial = double(trecho1);    // sintaxe do C++
double total = (double) viagem;      // sintaxe do C

// conversão implícita
double maior = trecho2;
```

Funções de Conversão

- Como criar funções de conversão?
  - Usando o seguinte padrão

```
operator tipo();
```

  - E obedecendo as regras:
    - Deve ser um método da classe
    - Não deve ter tipo de retorno
    - Não deve ter parâmetros

A **classe** define o tipo de origem e nome da função o tipo de destino

```
// funções de conversão
operator double();
operator int();
```

Funções de Conversão

- Classe Tempo atualizada

```
class Tempo
{
private:
   int horas;
   int minutos;

public:
   Tempo(int h = 0, int m = 0);

   operator double();
   operator int();
   ...
};
```

```
#include "Tempo.h"

Tempo::operator double()
{
    return horas + minutos / 60.0;
}

Tempo::operator int()
{
    return horas;
}
```

Funções de Conversão

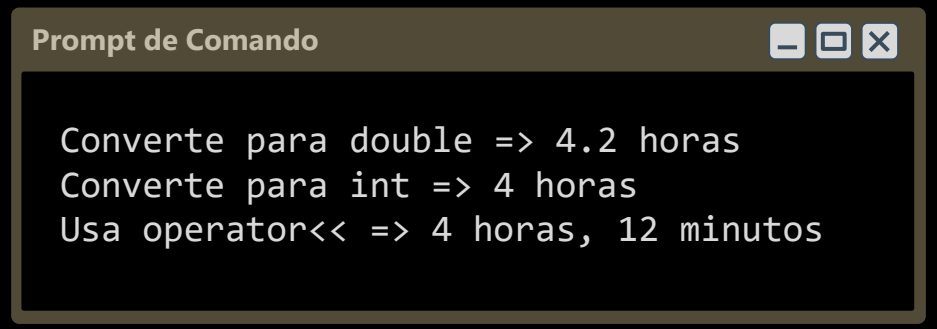
- Exemplo:

```
#include <iostream>
#include "Tempo.h"

int main()
{
   Tempo t { 4, 12 };

   double horas = t;    // conversão implícita

   std::cout << "Converte para double => " << horas << " horas\n";
   std::cout << "Converte para int => " << int(t) << " horas\n";
   std::cout << "Usa operator<< => " << t << "\n";
}
```



Problemas em Conversões

- **Conversões implícitas** podem ser problemáticas
  - Elas podem acontecer quando você não espera

```
Tempo i { 1, 10 };          // nome ruim... mas possível
...

int j = 0;
int vet[10];
...

cout << vet[i] << "\n";    // usou i em vez de j
```

Conversão de Tempo em int

Problemas em Conversões

- É **mais seguro** usar apenas conversões explícitas
  - É possível desligar as implícitas

```
class Tempo
{
    ...

    explicit operator double();

    explicit operator int();

    ...
};
```

```
int main()
{
    Tempo t { 3, 20 };

    int a = int(t);      ✓  // explícita
    int b = (int) t;     ✓  // explícita
    int c = t;           ✗  // implícita
}
```

Resumo

- Conversões de tipos para **classes**:
  - Construtor: converte o tipo usado no parâmetro para o tipo da classe

```
Tempo(int h);  // converte int em Tempo
```

  - Função de conversão: converte um objeto da classe em algum outro tipo

```
operator int();  // converte Tempo em int
```

  - Conversões implícitas: podem ser inibidas com `explicit`