Programação Orientada a Objetos introdução a classes

- Definição de classe
- Exemplos de classe
- Objetos
- Construtores
- Destrutores

Classe

- A classe é o Pilar de C ++
  - Possibilita encapsulamento, ocultação de dados e herança
- Tipo
  - Representação concreta de um conceito
    - Por exemplo. **float** com operações como: -, *, +
- Classe
  - Um tipo definido pelo usuário
  - Consiste em dados e métodos
  - Define propriedades e comportamento

Classe

- Vantagens
  - Tipos correspondem aos conceitos do programa
  - Programa conciso
  - Análise de código mais fácil
  - Compilador pode detectar usos ilegais dos tipos

- Abstração de dados
  - Separa os detalhes de implementação de suas propriedades essenciais

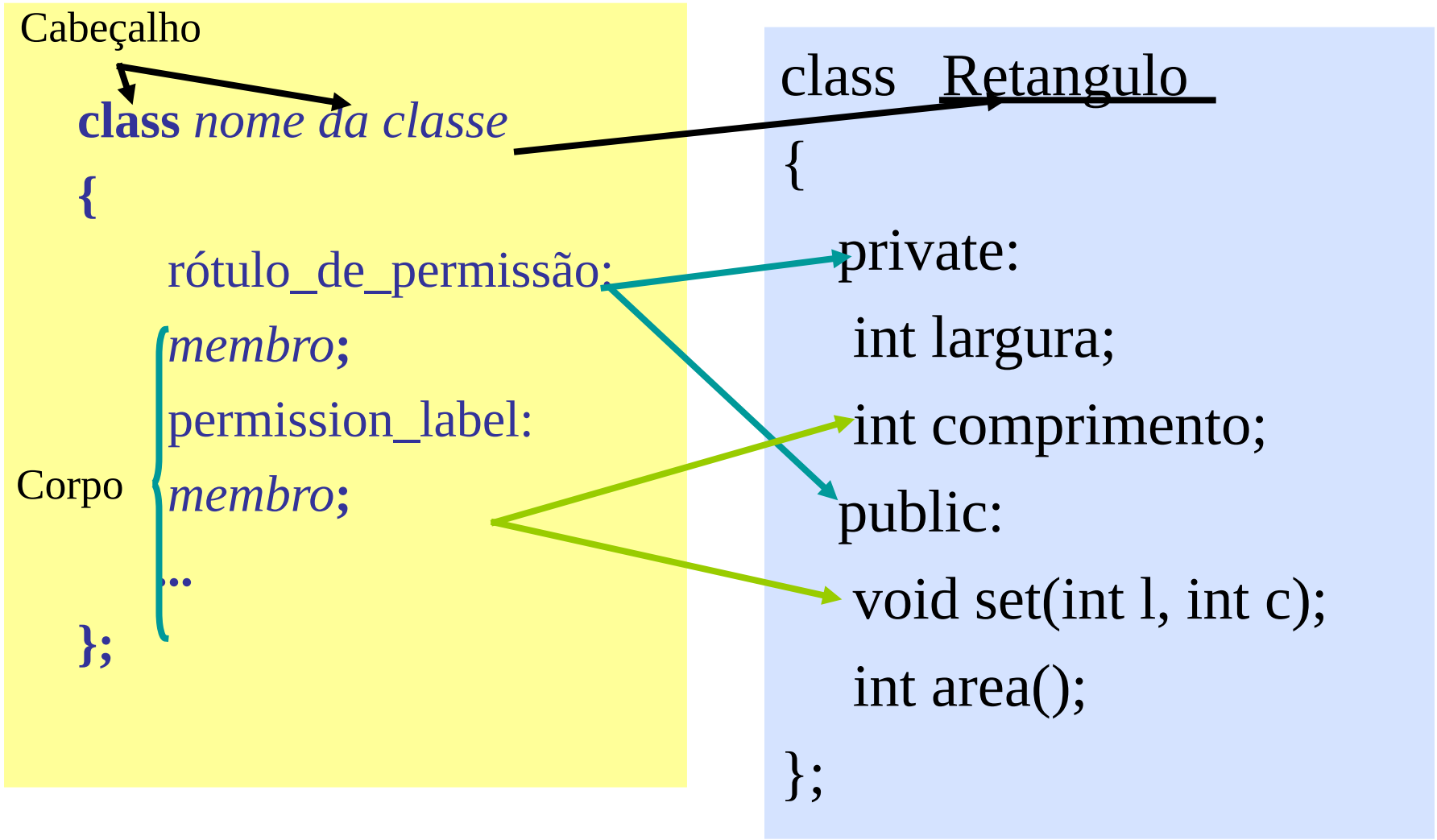
Aulas e Objetos

```
class Retangulo {
    private:
     int largura;
     int comprimento;
    public:
     void set(int l, int c);
     int area();
};
```

objetos: Instância de uma classe

```
Retangulo r1;
Retangulo r2;
Retangulo r3;
```

Definir um tipo de classe



Exemplo 1

```
#include <iostream>
using namespace std;
class Retangulo {
    private:
     int largura;
     int comprimento;
    public:
     void set(int l, int c);
     int area() { // primeira forma de implementar um metodo
        return this->largura * this->comprimento;
     }
};
```

Exemplo 1

```
void Retangulo::set(int x, int y) {
// outra forma de implementar um metodo
  largura = x;
  comprimento = y;
}
int main() {
    Retangulo ret;
    int x, y;
    cout << "Digite a largura e a altura de um retangulo: ";
    cin >> x >> y;

    ret.set(x,y);
    cout << "Area do retangulo: " << ret.area();
}
```

Exercício 1

- Escreva uma classe para representar Lampadas respeitando os seguintes requisitos:
- a. Para cada Lampada, deve-se armazenar seu tipo em uma variável char (ex: Incandescente - i, Fluorescente - f, Led - l) e potência em watts em uma variável int (ex: 100, 40, 9, ...).
- b. Elabore uma função main que deverá permitir que o usuário digite o tipo e a potência de uma lâmpada e armazene estas informações em um objeto da classe Lampada.
- c. Modifique a classe Lampada criando uma operação chamada mostraDados que exiba os atributos de uma lampada no console.
- d. Modifique a função main para que seja invocada a operação mostraDados antes do término do programa para exibir os dados armazenados no objeto da classe Lampada.

Definição de classe
Funções-membro

- **const** função membro
  - declaração
    - *tipo_de_retorno* (*lista_de_parâmetros*) const;
  - definição
    - *tipo_de_retorno* (*lista_de_parâmetros*) const {...}
    - *tipo_de_retorno nome_da_classe* :: *nome_da_função* (*lista_de_parâmetros*) const {...}
  - Não faz nenhuma modificação sobre os dados membros (função de segurança)
  - É ilegal uma função const modificar um dado membro da classe

Função Constante

```
class Tempo
{
  private:
    int h, min, s;
  public:
    void Escreva( ) const;
};
```

declaração da função

definição de função

```
void Tempo :: Escreva () const
{
    cout << hrs << “:” << mins << “:” << segundos
<< endl;
}
```

Exercício 2

- Modifique o exercício anterior de forma que o método mostraDados seja const.

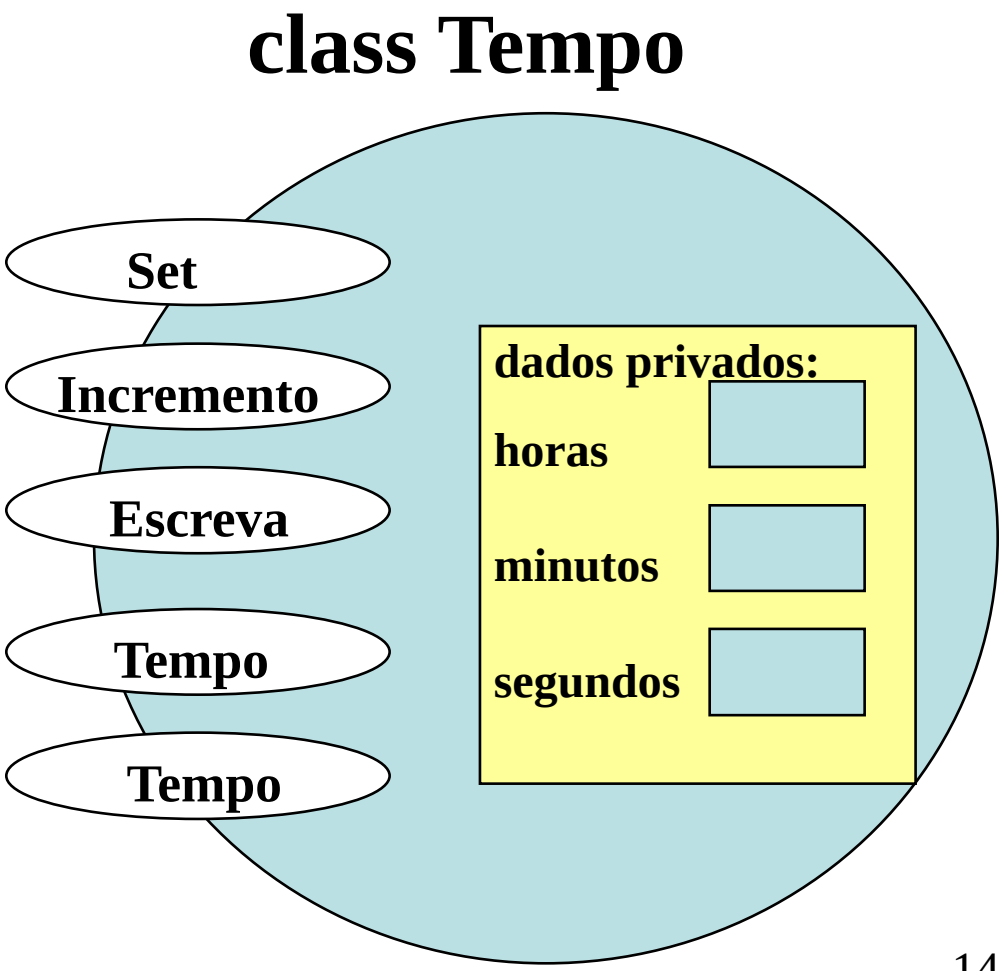
Definição de Classe - Controle de Acesso

- Ocultação de informações
  - Para evitar a representação interna de acesso direto de fora da classe
- Especificadores de acesso
  - public
    - pode ser acessível a partir de qualquer lugar dentro de um programa
  - private
    - pode ser acessado apenas por as funções membro e amigos desta classe
  - protected
    - atua como pública para classes derivadas
    - se comporta como privado para o resto do programa

Especificação class Tempo

```
class Tempo {

  public:

    void set ( int horas,  int minutos,  int segundos);
    void      Incremento ();
    void      Escreva ( ) const;
    Tempo (int initHrs, int initMins, int initSecs); // construtor
    Tempo ( );                  // construtor padrão

  private:

    int horas;
    int minutos;
    int segundos ;
};
```

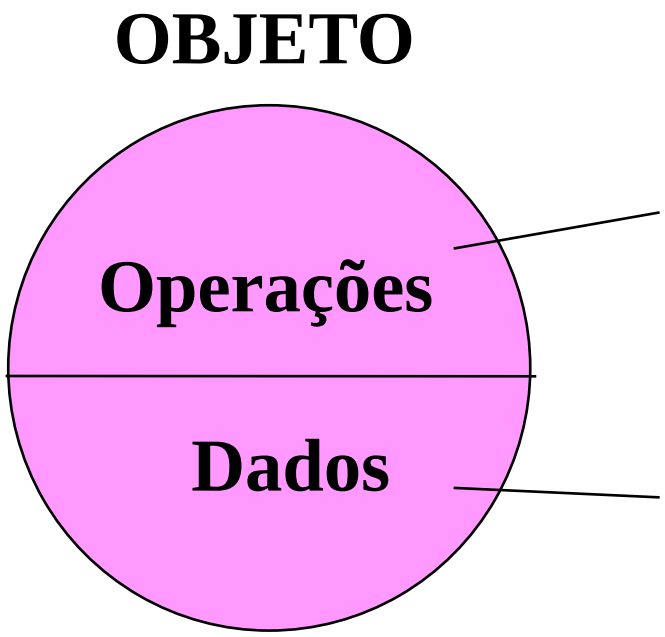
Classe Diagrama de interface



Definição de classe
Controle de acesso

- O especificador de acesso padrão é *private*
- Os dados são geralmente private ou protected
- Um método **private** é um auxiliar, só podem ser acessados por uma outra função da mesma classe (exceto classes amigas – não será abordado agora)
- As funções do tipo **public** fazem parte da *interface da classe*
- Cada seção de controle de acesso é opcional, repetíveis e seções podem ocorrer em qualquer ordem

O que é um objeto?



um conjunto de métodos (funções)

Estado interno
(valores de membros de dados privados)

Declaração de um objeto

```
class Retangulo
{
    private:
     int largura;
     int comprimento;
    public:
     void set(int l, int c);
     int area ();
};
```

```
main()
{
    Retangulo r1;
    Retangulo r2;

    r1.set (5, 8);
    cout << r1.area () << endl;

    r2.set (8,10);
    cout << r2.area () << endl;
}
```

Outro exemplo

```
#include <iostream.h>

 class Circulo{
  private:
  double raio;
  public:
  void armazena(double);
  double area(void);
  void exibe(void);
};
```

```
// definições de funções membro
void circulo :: armazena
   (double r)
{
   raio = r;
}
double circulo :: area (void)
{
   return 3,14 * raio * raio;
}
void circulo :: exibe (void)
{
   cout << “r = “<< raio << endl;
}
```

```
int main (void) {
    Circulo c; // um objecto da classe círculo
    c.armazena(5.0);
    cout << "A área do círculo c é " << c.area() << endl;
    c.exibe();
}
```

Declaração de um objeto – operador .

```
class Retangulo {
    private:
     int largura;
     int comprimento;
    public:
     void set(int l, int c);
     int area ();
};
```

r1 é alocado estaticamente

```
main()
{
➡ Retangulo r1;

➡ r1.set (5, 8);
}
```

r1

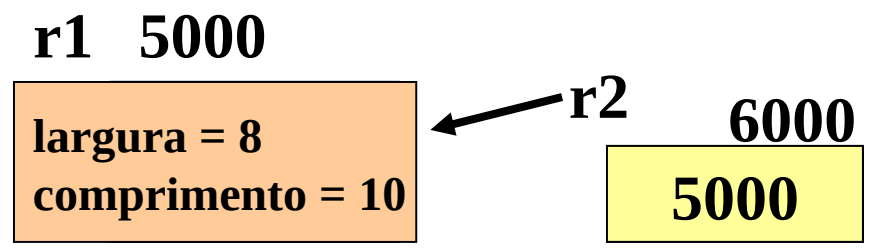
largura = 5
comprimento = 8

Declaração de um objeto – operador ->

```
class Retangulo
{
   private:
    int largura;
    int comprimento;
   public:
    void set(int l, int c);
    int area ();
};
```

r2 é um ponteiro para um objecto retângulo

```
main()
{
    Retangulo R1;
    r1.set (5, 8);     //notação de ponto

    Retangulo *r2;
    r2 = &r1;
    r2-> set(8,10);  // notação seta
}
```



Declaração de um objeto

```
class Retangulo
{
    private:
     int largura;
     int comprimento;
    public:
     void set(int l, int c);
     int area();
};
```

R3 é alocada dinamicamente

```
main()
{
     Retangulo *r3;
     r3 = new Retangulo();

     r3->set(80, 100); // notação seta

     delete r3;
     r3 = NULL;
}
```

r3 **6000**

NULO

Exercício 3

- Modifique o exercício anterior de forma que a utilizar alocação dinâmica de memória e o operador -> para criar e manipular objetos da classe Lampada.

Construtores

- Um objeto pode ser inicializado por um construtor de classe
  - construtor padrão
  - construtor de cópia
  - construtor com parâmetros
- Os recursos são alocados quando um objeto é inicializado
- Os recursos devem ser revogada quando um objeto está prestes a terminar a sua vida útil

Inicialização objeto

```
#include <iostream.h>

class Circulo {
  public:
  double raio;
};
```

1. Por atribuição

- Só funciona para dados públicos membros
- Nenhum controle sobre as operações nos dados

```
int main ()
{
        Circulo c1;          // Declare uma instância da classe círculo
   c1.raio = 5;              // Inicializar pela atribuição

}
```

Inicialização objeto

```
#include <iostream.h>
 class Circulo
{
  private:
   double raio;
  public:
   void setRaio(double r){
        raio = r;
   }
   double getRaio()
        {return raio;}
};
```

2. Por métodos públicos

```
int main (void) {
    circulo c;
    c.set Raio(5.0);
    cout << "O raio do círculo c
    é" << c.getRaio () << endl;
    // acesso de um membro de //
    dados privado com um
   // assessor público
}
```

Inicialização do objeto

```
class Retangulo
{
    private:
     int largura;
     int comprimento;
    public:
     Retangulo();
     Retangulo (const
    Retangulo &r);
     Retangulo (int l, int c);
     void set(int l, int c);
     int area ();
}
```

3. Pelo Construtor

- construtor padrão
- construtor de cópia
- Construtor com parâmetros

Eles são acessíveis ao público

Tem o mesmo nome que a classe

Não há nenhum tipo de retorno

São usados para inicializar dados membros da classe

Eles têm diferentes assinaturas

Inicialização do objeto

```
class Retangulo
{
    private:
     int largura;
     int comprimento;
    public:
     void set(int l, int c);
     int area ();
};
```

Quando uma classe é declarada sem nenhum construtor, o compilador assume automaticamente um construtor padrão e de cópia para ele.

- Construtor padrão

```
Retangulo :: Retangulo () {};
```

- Construtor de cópia

```
Retangulo :: Retangulo (const
    Retangulo &r)
{
    largura = r.largura;
    comprimento = r.comprimento;
};
```

Inicialização do objeto

```
class Retangulo
{
    private:
     int largura;
     int comprimento;
    public:
     void set(int l, int c);
     int area ();
}
```

- Inicializar com construtor padrão

```
Retangulo r1;

Retangulo *r3 = new Retangulo();
```

- Inicializar com construtor de cópia

```
Retangulo r4;
r4.set (60,80);

Retangulo r5 = r4;
Retangulo r6 (r4);

Retangulo *r7 = new Retangulo (r4);
```

Inicialização do objeto

```
class Rectangle
{
    private:
     int largura;
     int length;
    public:
     Retangulo (int l, int c){
      largura = l;
      comprimento = c;
    }
     void set(int l, int c);
     int area();
}
```

Se qualquer construtor com qualquer número de parâmetros é declarado, não vai existir construtor padrão, a menos que você defina um.

```
Retangulo r4;   // erro
```

- Inicializar com construtor

```
Retangulo r5 (60,80);
Retangulo *r6 = new Retangulo (60,80);
```

Exercício 4

- No programa anterior, verifique o funcionamento do construtor cópia utilizando o mesmo.

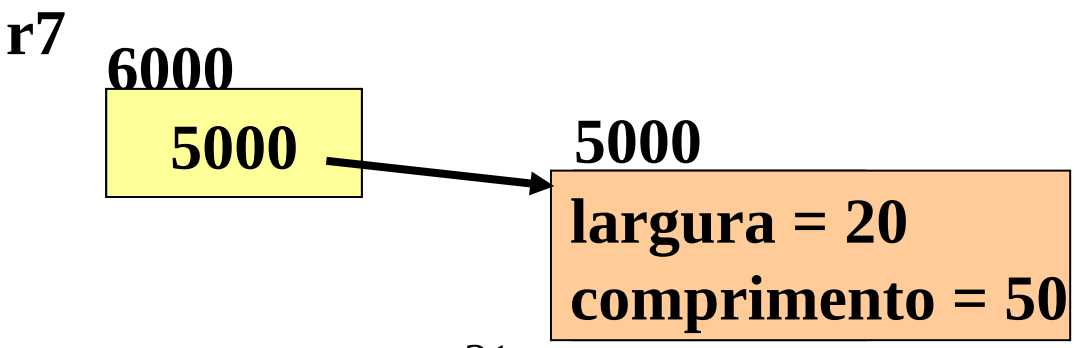
Inicialização

```
class Retangulo
{
     private:
      int largura;
      int comprimento;
     public:
      Retangulo();
      Retangulo (int l, int c);
      void set(int l, int c);
      int area();
}
```

Escrever seus próprios construtores

```
Retangulo :: Retangulo ()
{
    largura = 20;
    comprimento = 50;
};
```

```
Retangulo * r7 = new Retangulo();
```



Inicialização

Com construtores, temos mais controle sobre os membros de dados

```
class Conta
{
    private:
     char *nome;
     double balanco;
     unsigned int id;
    public:
     Conta();
     Conta (const Conta &c);
     Conta (const char *pessoa);
}
```

```
Conta :: Conta ()
{
    nome = NULL; balanco = 0.0;
    id = 0;
};
```

```
Conta :: Conta (const Conta &c)
{
    nome = new char [strlen (c.nome)
    + 1];
    strcpy (nome, c.nome);
    saldo = c.balanco;
    id = c.id;
};
```

```
Conta :: Conta (char const * pessoa)
{
    nome = new char [strlen (pessoa)
    1];
    strcpy (nome, pessoa);
    balanco = 0,0;
    id = 0;
};
```

Exercício 5

- Crie um construtor para a classe Lâmpada, que receba o tipo de lâmpada, e a potência como argumentos e inicialize os atributos da Lampada. Utilize o mesmo na função principal.

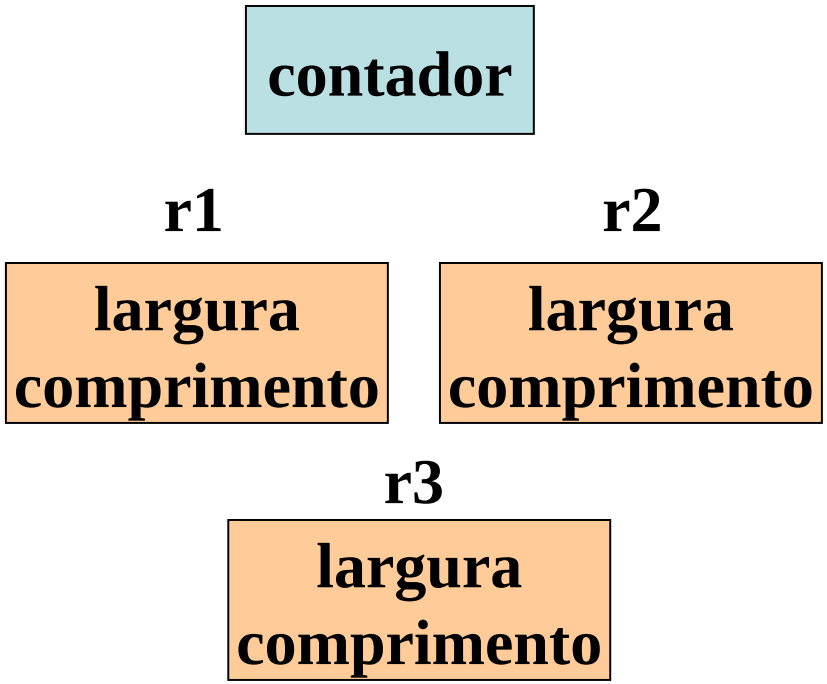
Definição de classe
Membros de dados

- Pode ser de qualquer tipo definido pelo usuário
- Membro de dados *não estáticos*
  - Cada objeto de classe tem sua própria cópia
- Membro de dados *estáticos*
  - Atua como uma variável global
  - Uma cópia por tipo de classe, por exemplo, contador.

Membro de dados estáticos

```
class Retangulo
{
   private:
    int largura;
    int comprimento;
    static int contador;
   public:
    void set(int l, int c);
    int area();
}
```

```
Retangulo r1;
Retangulo r2;
Retangulo r3;
```



Limpeza de um objeto - Destrutor

```
class Conta
{
    private:
     char *nome = new char[30];
     double balanco;
     unsigned int id; //único
    public:
     Conta();
     Conta (const Conta &a);
     Conta (const char *pessoa);
     ~ Conta();
}
```

Destrutor

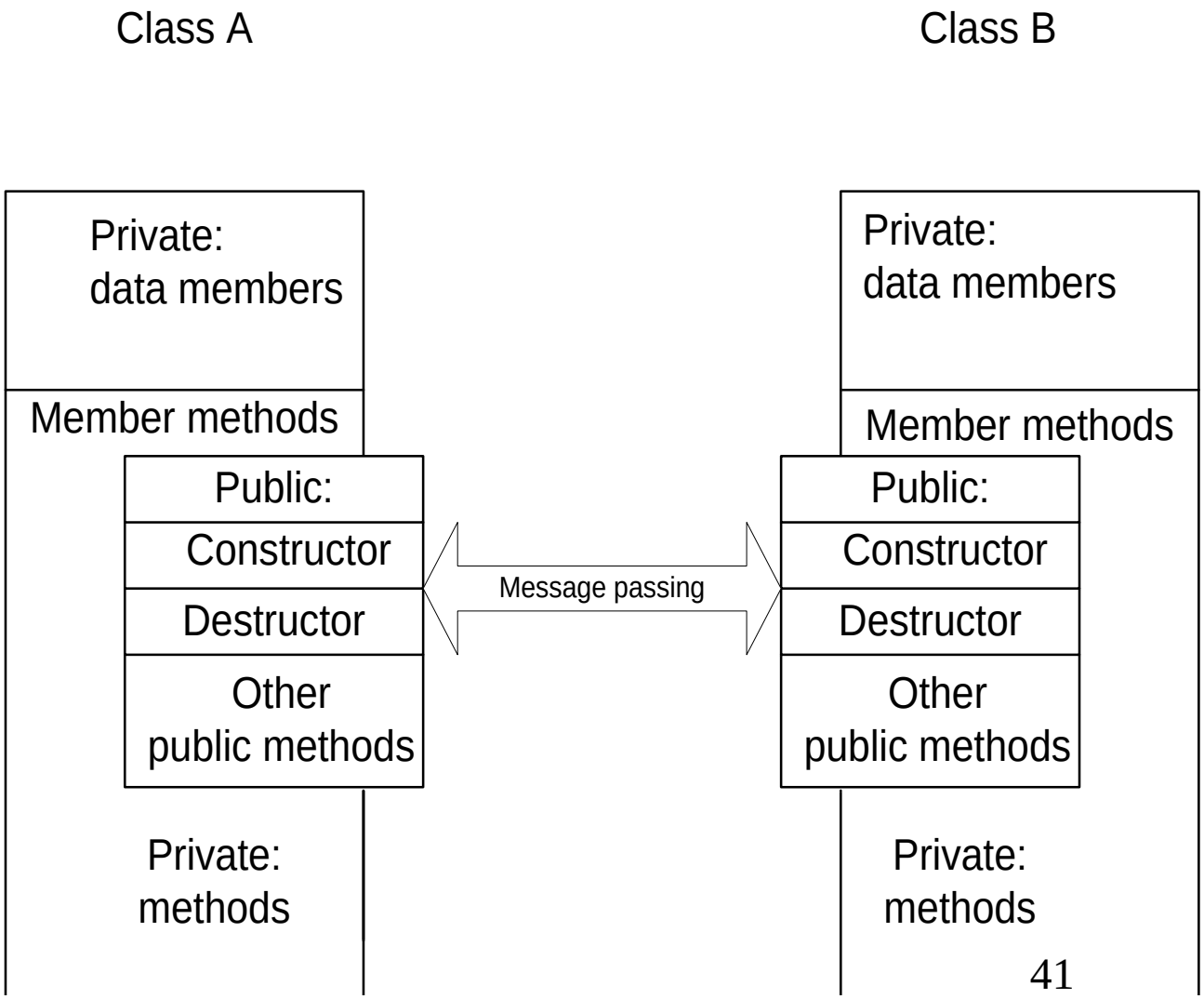
```
Conta :: ~Conta ()
{
    delete[] nome;
}
```

- Seu nome é o nome da classe precedido por um ~ (til)
- Não tem nenhum argumento
- Ele é usado para liberar a memória alocada dinamicamente e realizar outras atividades de "limpeza"
- Ele é executado automaticamente quando o objeto sai do escopo

Exercício 6

- Crie um destrutor para a classe Lâmpada, e o utilize na função main.

Interação de objetos



Trabalhando com vários arquivos

- Para melhorar a legibilidade, manutenção e reutilização, os códigos são organizados em módulos.
- Ao trabalhar com códigos complicados,
  - Um par de arquivos .cpp e .h para cada grupo de classe
    - ficheiro.h contém o protótipo da classe
    - .cpp contém a definição / implementação da classe
  - Um arquivo .cpp que contém main() função, deve incluir todos os correspondentes .h arquivos onde as funções usadas no arquivo .cpp são definidas

Exemplo: tempo.h

```
// Especificação de arquivo (tempo.h)
// Especifica os membros de dados e
// funções membro protótipos.

#ifndef  _TIME_H
#define _TIME_H

class Tempo
  {
    public:
          . . .

    private:
          . . .
   };

#endif
```

Exemplo: Tempo.cpp

```
// IMPLEMENTAÇÃO ARQUIVO    (Tempo.cpp)
// Implementa as funções da classe Tempo


#include <iostream.h>
#include “tempo.h”// também deve aparecer no código do cliente
   ... ...
bool Tempo :: Equal ( Tempo outroTempo ) const

//      função valor == true, Se esse tempo for igual outroTime
//                  == false, caso contrário
{
    return ((horas) == outroTempo.horas)
            && (minutos == outroTempo.minutos)
```

Exemplo: Main.cpp

```
// Código do cliente (Main.cpp)
#include “tempo.h”

// outras funções, se houver

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

Compilar e executar:

g ++ -o mainExec *main.cpp tempo.cpp*

Compilação e Vinculação de arquivos separados