Programação Orientada a Objetos

Giselle Lopes Ferrari Ronque

ferrari@eletrica.ufpr.br

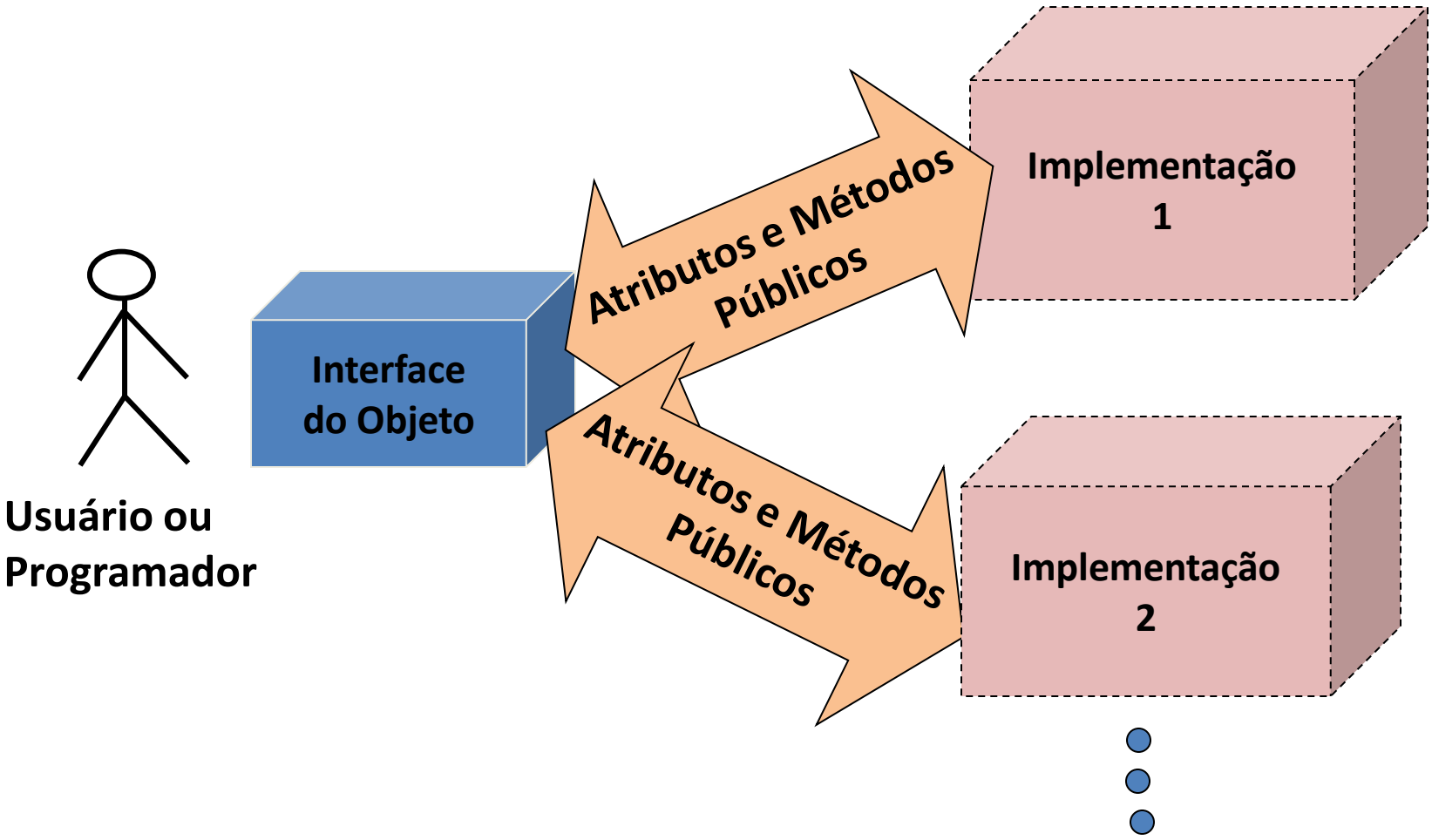
Sobrecarga

- As linguagens POO permitem que operadores e funções tenham usos múltiplos - **Sobrecarga**
- A sobrecarga possibilita maior flexibilidade no uso dos objetos. Exs:

Operador	**Uso I**	**Uso II**
<<	**Stream de saída:**	**Desloc. à esq.:**
	cout << “Alô Mundo!”	**x = y<<2**

Função	**Parâm. I**	**Parâm. II**	**Parâm. III**
imprime( )	**imprime(char c)**	**imprime(int n)**	**imprime(linha l, int j, char *s)**

Sobrecarga


Usuário ou
Programador
Interface
do Objeto
Atributos e Métodos
Públicos
Implementação
1
Atributos e Métodos
Públicos
Implementação
2

Sobrecarga de Operadores

- Sobrecarregar um operador significa redefinir seu símbolo, de maneira que ele se aplique também a tipos de dados definidos pelo usuário como classes e estruturas.
- A implementação de sobrecarga de operadores é definida por meio de funções chamadas **operadoras** , criadas como membro da classe ou como funções globais.

Sobrecarga de Operadores

- **LIMITAÇÕES**:
  - Respeitar a definição original do operador;
    - Ex: Não é possível mudar o operador de unário para binário.
  - Não se pode inventar símbolos novos;
    - Ex: ##
  - Não é possível modificar a precedência de operadores por meio de sobrecarga;
    - Ex: Operador de multiplicação (*) tem precedência sobre o operador de adição (+).
  - Operadores que não podem ser sobrecarregados:
    - . → operador ponto
    - :: → operador de resolução de escopo
    - ? : →operador ternário

Operadores

- Unários
  - Incremento ++
  - Decremento --

- Binários
  - + , - , * , / , > , += , etc.

Exemplo

```
#include<iostream.h>
#include<stdlib.h>
class ponto
{
    private:
        int x, y;
    public:
        ponto (int x1 = 0, int y1 = 0)
    {
            x=x1;
            y=y1;
        } // construtor
        void operator ++ ( ) {
            ++x;
            ++y;
        } // função operadora pré-fixada
        void printpt( ) {
        cout<< "(" << x << "," << y << ")";
        } // imprime ponto
} ; // Final da Classe
```

Exemplo

```
void main() {
    ponto p1, p2(2,3), p3;                  // declara e inicializa
    cout << "\n p1 = " ; p1.printpt( );      // imprime
    cout << "\n p2 = " ; p2.printpt( );      // imprime
    ++p1;                                   // incrementa p1
    ++p2;                                   // incrementa p2
    cout << "\n++p1 = " ; p1.printpt( );     // imprime
    p3 = p1;
    cout << "\n p3 = "; p3.printpt( );       // imprime
}
```

Exemplo

```
void main() {
    ponto p1, p2(2,3), p3;                  // declara e inicializa
    cout << "\n p1 = " ; p1.printpt( );      // imprime
    cout << "\n p2 = " ; p2.printpt( );      // imprime
    ++p1;                                   // incrementa p1
    ++p2;                                   // incrementa p2
    cout << "\n++p1 = " ; p1.printpt( );     // imprime
    p3 = p1;
    cout << "\n p3 = "; p3.printpt( );       // imprime
}
```

SAÍDA:

p1 = (0,0)
p2 = (2,3)
++p1 = (1,1)
p3 = (1,1)

Sobrecarga de Operadores Unários

- Incremento **PRÉ-FIXADA**

```
void operator ++ ( ) {
    ++x;
    ++y;
  }
```

- **++p1;**
- **p1.operator++();**

Exemplo

- p3 = ++p1; **// ERRO!!!**

```
ponto operator ++ ( ) {
        ++x;   ++y;
        ponto temp;
        temp.x = x;
        temp.y = y;
        return temp;
    } // função operadora pré-fixada
```

Sobrecarga de Operadores Unários

- Incremento **PÓS-FIXADO**
  - operator ++() → operação **pré-fixada**
  - operator ++(int) → operação **pós-fixada**
    - **Função ignora o parâmetro tipo *int* e serve somente para que o compilador possa diferenciar as duas formas de uso.**

*** A operação pré ou pós-fixada deve ser implemetada pelo programador.**

*** O compilador não reconhece automaticamente a operação PRÉ ou PÓS-fixada**

Exemplo

```
#include<iostream.h>
#include<stdlib.h>
class ponto
{
    ...
    ponto operator ++ ( ) {
            ++x;      ++y;
            ponto temp;
            temp.x = x;
            temp.y = y;
            return temp;
    } // função operadora pré-fixada
```

```
    ponto operator ++ ( int ) {
            ++x;      ++y;
            ponto temp;
            temp.x = x - 1;
            temp.y = y - 1;
            return temp;
    } // função operadora pós-fixada
    ...
} ; // Final da Classe
```

Exemplo

```
void main() {
    ponto p1, p2(2,3), p3;                    // declara e inicializa
    cout << "\n p1 = " ; p1.printpt( );        // imprime
    cout << "\n p2 = " ; p2.printpt( );        // imprime
    cout << "\n ++p1 = " ; (++p1).printpt( );  // incrementa e imprime
    cout << "\n p2++ = " ; (p2++).printpt( );  // imprime e incrementa
    cout << "\n p2 = " ; p2.printpt( );        // imprime
    p3 = p1++;                                // atribui e incrementa
    cout << "\n p3 = "; p3.printpt( );         // imprime
    cout << "\n p1 = "; p1.printpt( );         // imprime

}
```

Exemplo

```
void main() {
    ponto p1, p2(2,3), p3;                  // declara e inicializa
    cout << "\n p1 = " ; p1.printpt( );      // imprime
    cout << "\n p2 = " ; p2.printpt( );      // imprime
    cout << "\n ++p1 = " ; (++p1).printpt( ); // incrementa e imprime
    cout << "\n p2++ = " ; (p2++).printpt( ); // imprime e incrementa
    cout << "\n p2 = " ; p2.printpt( );      // imprime
    p3 = p1++;                              // atribui e incrementa
    cout << "\n p3 = "; p3.printpt( );       // imprime
    cout << "\n p1 = "; p1.printpt( );       // imprime

}
```

SAÍDA:

p1 = (0,0)
p2 = (2,3)
++p1 = (1,1)
p2++ = (2,3)
p2 = (3,4)
p3 = (1,1)
p1 = (2,2)

Sobrecarga de Operadores Binários

- Função operadora tem argumentos;
  - Sempre necessita de um argumento a menos do que o número de operandos daquele operador;

Exemplo

```
#include<iostream.h>
class ponto
{
    private:
        int x, y;
    public:
    ponto  (int x1 = 0, int y1 = 0) {
        x = x1; y = y1;
    } // construtor
    ponto operator +(ponto p) {
        ponto temp;
        temp.x = x + p.x;
        temp.y = y + p.y;
        return temp;
    } // soma dois objetos
    ponto operator +(int n) {
        ponto temp;
        temp.x = x + n;
        temp.y = y + n;
        return temp;
    } ; // soma um número a um objeto
    void printpt() {
        cout<< "(" << x << "," << y << ")";
    } // imprime ponto
};
```

Exemplo

```
void main() {
    ponto p1(5,1), p2(2,3), p3;
    cout << "\n p1 = " ; p1.printpt( );
    cout << "\n p2 = " ; p2.printpt( );
    p3 = p1 + p2;                          //p3 = p1.operator+(p2);
    cout << "\n p3 = " ; p3.printpt( );
    p3 = p1 + 5;                           //p3 = p1.operator+(5);
    cout << "\n p3 = "; p3.printpt( );
}
```

Exemplo

```
void main() {
    ponto p1(5,1), p2(2,3), p3;
    cout << "\n p1 = " ; p1.printpt( );
    cout << "\n p2 = " ; p2.printpt( );
    p3 = p1 + p2;                    //p3 = p1.operator+(p2);
    cout << "\n p3 = " ; p3.printpt( );
    p3 = p1 + 5;                     //p3 = p1.operator+(5);
    cout << "\n p3 = "; p3.printpt( );
}
```

SAÍDA:

p1 = (5,1)
p2 = (2,3)
p3 = (7,4)
p3 = (10,6)

Exercício 1

- Defina uma classe chamada **complexo**. Esta classe deve armazenar a parte real e imaginária de um número complexo (variáveis do tipo *float*). Inclua:
  - Dois construtores: o primeiro sem argumentos e o outro que recebe a parte real e imaginária do número complexo;
  - Uma função membro que retorna a parte real e outra que retorna a parte imaginária do número complexo;
  - Os operadores: adição (+), subtração (-) , multiplicação (*), divisão (/); *****
  - As operações aritméticas de atribuição(+= , -= , *= , /=); *****
  - As operações relacionais (>, >=, < , <= , == , !=);*****
  - Criar um **programa principal** para testar a classe.

***Sobrecarga**

Exercício 2

- Defina uma classe chamada **String**.
- A classe deve permitir comparações (<, >, ==) e concatenações (+ , +=). *
- Criar um **programa principal** para testar a classe.

***Sobrecarga**

Exercício 3

- Defina uma classe chamada **fração**. Esta classe deve armazenar o numerador e o denominador de uma fração em duas variáveis inteiras. Inclua:
  - Dois construtores: o primeiro sem argumentos e o outro que recebe numerador e denominador da fração;
  - Uma função membro que retorna o numerador e outra que retorna o denominador;
  - As operações de adição, subtração, multiplicação e divisão;*****
  - As operações de incremento e decremento pré e pós-fixadas; *****
  - Criar um **programa principal** para testar a classe.

***Sobrecarga**