Polimorfismo de Sobrecarga e Parâmetros default em C++

EDUARDO HABIB BECHELANE MAIA

HABIB@CEFETMG.BR

Polimorfismo de Sobrecarga

Permite que um método de determinado nome tenha comportamentos distintos, em função de diferentes parâmetros que ele recebe.

Cada método difere:

No **número**	No **tipo** de **parâmetros**.

Exemplo em C++

```
// Forma.hpp
#ifndef FORMA_HPP
#define FORMA_HPP

class Forma {
public:
    double calcularArea(double raio);
    double calcularArea(double comprimento, double largura);
};

#endif //FORMA_HPP
```

Forma.cpp

```
// Forma.cpp
#include "Forma.hpp"

// Sobrecarga para calcular a área de um círculo
double Forma::calcularArea(double raio) {
    return 3.14159 * raio * raio;
}

// Sobrecarga para calcular a área de um retângulo
double Forma::calcularArea(double comprimento, double largura) {
    return comprimento * largura;
}
```

Main.cpp

```cpp
// main.cpp
#include <iostream>
#include "Forma.hpp"

int main() {
    Forma forma;
    std::cout << "Área do círculo: " << forma.calcularArea(5.0) << std::endl;
    std::cout << "Área do retângulo: " << forma.calcularArea(5.0, 4.0) << std::endl;
    return 0;
}
```

Código em java

```
public class Forma {

    // Sobrecarga para calcular a área de um círculo
    public double calcularArea(double raio) {
        return Math.PI * raio * raio;
    }

    // Sobrecarga para calcular a área de um retângulo
    public double calcularArea(double comprimento, double largura) {
        return comprimento * largura;
    }

    public static void main(String[] args) {
        Forma forma = new Forma();
        System.out.println("Área do círculo: " + forma.calcularArea(5.0));
        System.out.println("Área do retângulo: " + forma.calcularArea(5.0, 4.0));
    }
}
```

Polimorfismo de Sobrecarga

A sobrecarga e os construtores

- O polimorfismo de sobrecarga normalmente acontece sobre os métodos construtores
- É comum para uma classe ter várias maneiras de instanciá-la.

Polimorfismo de Sobrecarga

```
1  #ifndef __PESSOA_HPP
2  #define __PESSOA_HPP
3
4  #include <iostream>
5  using namespace std;
6
7  class Pessoa{
8      private:
9          string nome;
10         string cpf;
11     public:
12         Pessoa (string nome, string cpf);
13         Pessoa (string nome);
14         void setNome(string nome);
15         string getNome();
16         void setCpf(string cpf);
17         string getCpf(string cpf);
18 };
19
20 #endif
```

Polimorfismo de Sobrecarga

```cpp
#include <iostream>
#include "Pessoa.hpp"

using namespace std;

Pessoa::Pessoa(string nome, string cpf){
    this->nome = nome;
    this->cpf = cpf;
}

Pessoa::Pessoa(string nome){
    this->nome= nome;
    this->cpf = "";
}

```

```cpp
void Pessoa::setNome(string nome){
    this->nome = nome;
}

string Pessoa::getNome(){
    return this->nome;
}

void Pessoa::setCpf(string cpf){
    this->cpf = cpf;
}

string Pessoa::getCpf(string cpf){
    return this->cpf;
}
```

Valores default para os argumentos de uma função

Uma função pode ser chamada sem serem especificados nenhum de seus argumentos

O protótipo de uma função pode fornecer valores *default* para esses argumentos não especificados

Se forem omitidos os argumentos correspondentes na chamada à função, os valores *default* serão automaticamente usados;

Se forem especificados, estes serão respeitados e usados;

Matematica.hpp

```
#ifndef __MATEMATICA_HPP // se o header não está definido
#define __MATEMATICA_HPP // define o header
#include<iostream> //biblioteca
#include <string.h>

using namespace std;

class Matematica
{
        public:
          double quotient(int numerador=1,int denominador=1);

};
#endif
```

Matematica.cpp

```
#include <iostream>
using namespace std;
#include "Matematica.hpp"

double Matematica::quotient( int numerador, int denominador )
{
  // não pode dividir por 0
  if ( denominador == 0 )
        {
              cout << "Divisão por 0";
              return 0;
        }
   return (double) (numerador)  / denominador;
}
```

Valores default para os argumentos de uma função

Se um argumento for omitido, todos os subsequentes deverão sê-lo:

- resultado=mat.quotient(,4); // ERRADO
- resultado=mat.quotient(); // CERTO

Após a primeira especificação com valor default, todos os parâmetros seguintes devem ser especificados com valor default:

- double quotient(int numerador=1,int denominador); // ERRADO
- double quotient(int numerador=1,int denominador=1); // CERTO

Main.cpp

```
#include <iostream>
#include "Matematica.hpp"

int main(int argc, char** argv)
{
		Matematica mat;

		float resultado=mat.quotient();

		cout << "\nResultado 1: " << resultado;

		resultado=mat.quotient(2);

		cout << "\nResultado 2: " << resultado;

		resultado=mat.quotient(2,4);

		cout << "\nResultado 3: " << resultado;
		return 0;
}
```

Em Java

Em Java, não existe uma correspondência direta para os parâmetros default

- Pode-se obter um comportamento semelhante usando sobrecarga de métodos. Ex:

```
public class Matematica {

    // Método com dois parâmetros
    public double quotient(int numerador, int denominador) {
        return numerador / (double) denominador;
    }

    // Sobrecarga do método sem parâmetros
    public double quotient() {
        return quotient(1, 1);  // Valores default são usados aqui
    }

    // Sobrecarga do método com apenas o numerador
    public double quotient(int numerador) {
        return quotient(numerador, 1);  // Valor default para denominador
    }
}
```