CIV2801 – Fundamentos de Computação Gráfica Aplicada

2024.2

Introdução à Programação Orientada a Objetos

Luiz Fernando Martha

Orientacão a Objetos

A maioria dos métodos utilizados em ambientes de desenvolvimento de software se baseia em uma **decomposição funcional** e/ou **controlada por dados** dos sistemas. Estas abordagens se diferem em diversos aspectos das abordagens que adotam **metodologias orientadas a objetos**, onde **dados e funções são altamente integrados**.

O desenvolvimento de software com a abordagem orientada à objetos consiste na construção de **módulos independentes** ou **objetos** que podem ser **facilmente substituídos**, **modificados** e **reutilizados**. Ela retrata a visão do mundo real como um sistema de objetos cooperativos e colaborativos. Neste caso, o software é uma **coleção de objetos** discretos que **encapsulam dados** e **operações** executadas nesses dados para modelar objetos do mundo real. A **classe** descreve um grupo de objetos que têm estruturas semelhantes e operações similares.

A filosofia Orientada a Objetos é **muito similar ao mundo real** e, portanto, vem ganhando popularidade pois os sistemas aqui são vistos como um conjunto de objetos que interagem assim como no mundo real. Para implementar este conceito, a programação estruturada baseada em processos não é utilizada; em vez disso, os **objetos são criados usando estruturas de dados**. Assim como toda linguagem de programação oferece vários tipos de dados, da forma similar, no caso dos objetos certos tipos de dados são pré-definidos. (Nath, 2014 – Lecture Notes on Object-Oriented Methodology)

Orientacão a Objetos

A abordagem orientada a objetos possibilita uma melhor organização, versatilidade e reutilização do código fonte, o que facilita atualizações e melhorias nos programas. A abordagem orientada a objetos é caracterizada pelo uso de classes e objetos, e de outros conceitos que serão esclarecidos a seguir.

- **Classes** são espécies de montadoras de objetos, que definem suas características como, quais funções são capazes de realizar e quais os atributos que o objeto possui. Essa forma de programar permite ao usuário resolver problemas utilizando conceitos do mundo real.

- **Objeto** é uma instancia gerada a partir de uma classe. Um objeto é identificado a partir dos métodos e dos atributos que possui.

- **Encapsulamento** é o ato de esconder do usuário os processos internos de um objeto, classe ou método.

- **Herança** (e **Polimorfismo**) é uma característica que permite a determinada classe herdar as características de outra classe. Ou seja, a classe descendente adquiri todos os métodos e atributos da classe pai.

Métodos são as funções que objeto pode realizar.
Atributo é tudo que um objeto possui como variável.

Orientacão a Objetos

Classe, Objeto e Encapsulamento

```
#ifndef STACK_H
#define STACK_H



class Stack
{
public:
  Stack();
  ~Stack();
  void push(double _n);
  double pop();
  bool isEmpty();
  void show();

private:
  int m_top;
  double *m_elems;
};

#endif
```

```
#ifndef REAL_H
#define REAL_H



class Real
{
public:
  Real(double _val);
  ~Real();
  Real sum(Real _n);
  Real sub(Real _n);
  Real mul(Real _n);
  Real div(Real _n);

private:
  double m_value;
};


#endif
```

Orientacão a Objetos

Classe, Objeto e Encapsulamento

```
#ifndef STACK_H
#define STACK_H



class Stack
{
public:
  Stack();
  ~Stack();
  void push(double _n);
  double pop();
  bool isEmpty();
  void show();

private:
  int m_top;
  double *m_elems;
};

#endif
```

```
#ifndef STACK_H
#define STACK_H

#include "real.h"

class Stack
{
public:
  Stack();
  ~Stack();
  void push(Real _n);
  Real pop();
  bool isEmpty();
  void show();

private:
  int m_top;
  Real* m_elems;
};

#endif
```

```
#ifndef REAL_H
#define REAL_H



class Real
{
public:
  Real(double _val);
  ~Real();
  Real sum(Real _n);
  Real sub(Real _n);
  Real mul(Real _n);
  Real div(Real _n);

private:
  double m_value;
};


#endif
```

Orientacão a Objetos

A abordagem orientada a objetos possibilita uma melhor organização, versatilidade e reutilização do código fonte, o que facilita atualizações e melhorias nos programas. A abordagem orientada a objetos é caracterizada pelo uso de classes e objetos, e de outros conceitos que serão esclarecidos a seguir.

- **Classes** são espécies de montadoras de objetos, que definem suas características como, quais funções são capazes de realizar e quais os atributos que o objeto possui. Essa forma de programar permite ao usuário resolver problemas utilizando conceitos do mundo real.
- **Objeto** é uma instancia gerada a partir de uma classe. Um objeto é identificado a partir dos métodos e dos atributos que possui.
- **Encapsulamento** é o ato de esconder do usuário os processos internos de um objeto, classe ou método.
- **Herança** (e **Polimorfismo**) é uma característica que permite a determinada classe herdar as características de outra classe. Ou seja, a classe descendente adquiri todos os métodos e atributos da classe pai.

Métodos são as funções que objeto pode realizar.
Atributo é tudo que um objeto possui como variável.

Orientacão a Objetos

Herança e Polimorfismo

```
#ifndef REAL_H
#define REAL_H

class Real
{
public:
  Real(double _val);
  ~Real();
  Real sum(Real _n);
  Real sub(Real _n);
  Real mul(Real _n);
  Real div(Real _n);

private:
  double m_value;
};

#endif
```

```
#ifndef COMPLEX_H
#define COMPLEX_H

class Complex
{
public:
  Complex(double _re,
          double _im);
  ~Complex();
  Complex sum(Complex _n);
  Complex sub(Complex _n);
  Complex mul(Complex _n);
  Complex div(Complex _n);

private:
  double m_real;
  double m_imag;
};

#endif
```

Orientacão a Objetos

Herança e Polimorfismo

```
#ifndef REAL_H
#define REAL_H

class Real
{
public:
  Real(double _val);
  ~Real();
  Real sum(Real _n);
  Real sub(Real _n);
  Real mul(Real _n);
  Real div(Real _n);

private:
  double m_value;
};

#endif
```

```
#ifndef COMPLEX_H
#define COMPLEX_H

class Complex
{
public:
  Complex(double _re,
          double _im);
  ~Complex();
  Complex sum(Complex _n);
  Complex sub(Complex _n);
  Complex mul(Complex _n);
  Complex div(Complex _n);

private:
  double m_real;
  double m_imag;
};

#endif
```

```
#ifndef NUMBER_H
#define NUMBER_H

class Number
{
public:
  Number(int _type);
  ~Number();
  Number sum(Number _n);
  Number sub(Number _n);
  Number mul(Number _n);
  Number div(Number _n);

private:
  int m_type;
};

#endif
```

Orientacão a Objetos

Herança e Polimorfismo

```
#ifndef REAL_H
#define REAL_H

#include "number.h"

class Real : Number
{
public:
  Real(double _val);
  ~Real();
  Number sum(Number _n);
  Number sub(Number _n);
  Number mul(Number _n);
  Number div(Number _n);

private:
  double m_value;
};

#endif
```

```
#ifndef COMPLEX_H
#define COMPLEX_H

#include "number.h"

class Complex : Number
{
public:
  Complex(double _re,
          double _im);
  ~Complex();
  Number sum(Number _n);
  Number sub(Number _n);
  Number mul(Number _n);
  Number div(Number _n);

private:
  double m_real;
  double m_imag;
};

#endif
```

```
#ifndef NUMBER_H
#define NUMBER_H



class Number
{
public:
  Number(int _type);
  ~Number();
  Number sum(Number _n);
  Number sub(Number _n);
  Number mul(Number _n);
  Number div(Number _n);

protected:
  int m_type;
};

#endif
```

Orientacão a Objetos

Herança e Polimorfismo

```
#ifndef REAL_H
#define REAL_H

#include "number.h"

class Real : Number
{
public:
  Real(double _val);
  ~Real();
  Number sum(Number _n);
  Number sub(Number _n);
  Number mul(Number _n);
  Number div(Number _n);

private:
  double m_value;
};

#endif
```

```
#ifndef COMPLEX_H
#define COMPLEX_H

#include "number.h"

class Complex : Number
{
public:
  Complex(double _re,
          double _im);
  ~Complex();
  Number sum(Number _n);
  Number sub(Number _n);
  Number mul(Number _n);
  Number div(Number _n);

private:
  double m_real;
  double m_imag;
};

#endif
```

```
#ifndef NUMBER_H
#define NUMBER_H



class Number
{
public:
  Number(int _type);
  ~Number();
  virtual Number sum(Number _n) = 0;
  virtual Number sub(Number _n) = 0;
  virtual Number mul(Number _n) = 0;
  virtual Number div(Number _n) = 0;

protected:
  int m_type;
};

#endif
```